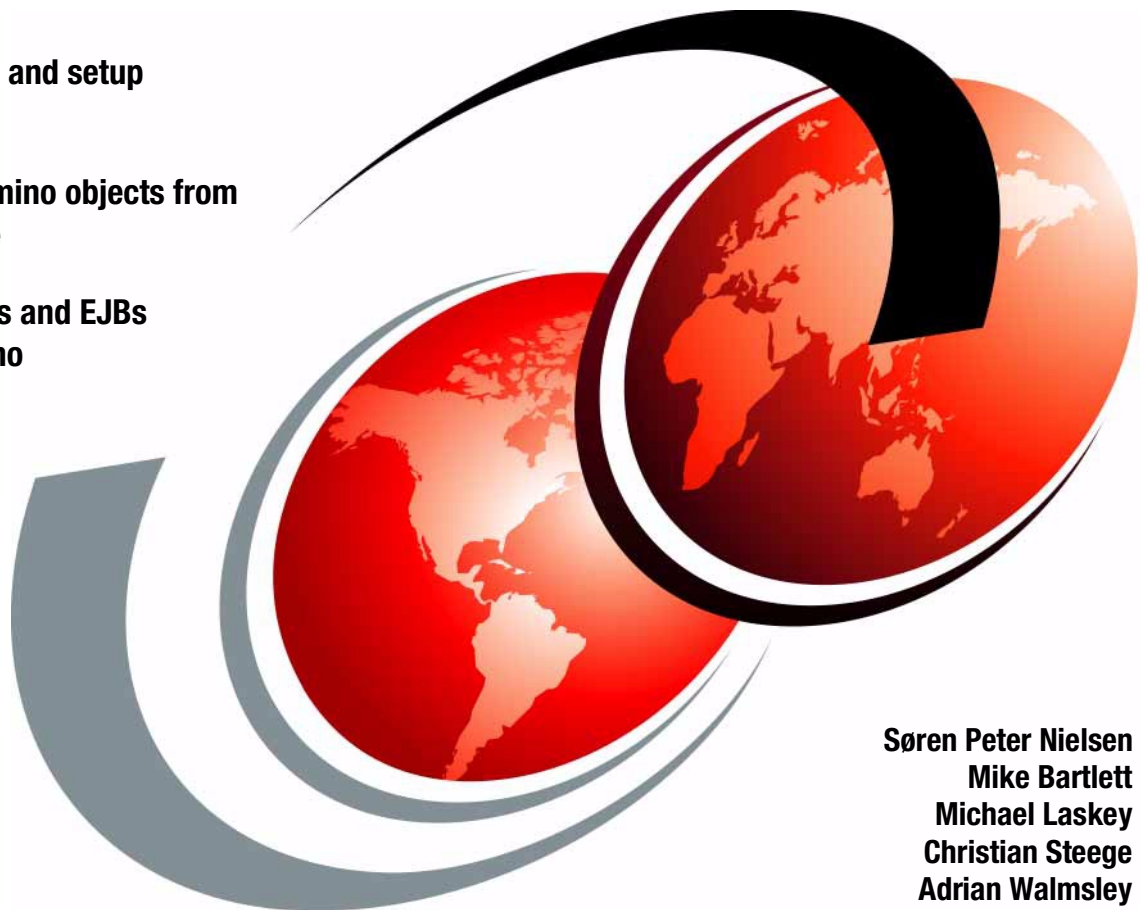


Domino and WebSphere Together

Installation and setup

Access Domino objects from WebSphere

Use servlets and EJBs from Domino



Søren Peter Nielsen
Mike Bartlett
Michael Laskey
Christian Steege
Adrian Walmsley

ibm.com/redbooks

Redbooks



International Technical Support Organization

Domino and WebSphere Together

August 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, "Special notices" on page 233.

First Edition (August 2000)

This edition applies to Lotus Domino R5.0.4 and IBM WebSphere Application Server Advanced Edition 3.02.1

This document created or updated on August 18, 2000.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. TQH Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000. All rights reserved.

Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	vii
The team that wrote this redbook	vii
Comments welcome	ix
 Chapter 1. The platform for collaborative commerce	1
1.1 The opportunity: Collaborative commerce	1
1.2 The problem: Collaboration on a global scale	1
1.3 The technology plan	2
1.4 Collaborative commerce with Domino and WebSphere	4
1.5 The IBM WebSphere e-business platform	5
1.6 Summary	6
 Chapter 2. Installation and setup	7
2.1 Introduction	7
2.2 Prerequisites	7
2.2.1 Platform	8
2.2.2 Product software levels	8
2.3 Creating a user with administration rights for DB2 and WebSphere ..	8
2.4 Installation of DB2 UDB Release 6.1 and fixpack	12
2.4.1 Installation of DB2 fixpack	17
2.5 Installing IBM JDK 1.1.7P	18
2.5.1 Editing the system CLASSPATH to include the JDK	20
2.6 Installing Domino Server R5	21
2.6.1 Domino Server configuration and setup	24
2.7 Installing WebSphere V3.02 and its service upgrade	30
2.7.1 Installing WebSphere Application Server V3.02	31
2.7.2 Installing the WebSphere 3.02 service upgrade	34
2.7.3 Starting and running WebSphere	36
2.7.4 Installing the WebSphere DSAPI plug-in for Domino	39
2.8 Using the IIS HTTP Web Server in place of Domino	45
2.9 Different Web server plug-in behavior	45
2.10 Summary	45
 Chapter 3. The elements of WebSphere applications	47
3.1 Java servlets	49
3.2 JavaServer Pages	50
3.3 Enterprise JavaBeans	52
3.3.1 Session beans	53
3.3.2 Entity beans	53
3.3.3 EJB architecture	54
3.4 Summary	57

Chapter 4. Accessing Domino R5 from WebSphere	59
4.1 Setting up IBM VisualAge for Java for the examples in this book	59
4.1.1 Installing VA Java	59
4.1.2 Adding features to VA Java	60
4.1.3 Importing the Domino R5 classes	61
4.1.4 Importing the IBM account example	65
4.1.5 Creating a new package for the example code	65
4.2 Setting up WebSphere to use Domino	65
4.2.1 Modifying the path WebSphere uses	66
4.3 Deploying the IBM WebSphere account example	67
4.3.1 Create a database for account objects	67
4.3.2 Install the database driver in WebSphere	67
4.3.3 Create a data source in WebSphere	69
4.3.4 Creating an EJB container	70
4.3.5 Create the Account and Transfer EJBs in WebSphere	71
4.3.6 Deploying the servlets and JSPs	73
4.4 Using JDBC to access Domino R5	75
4.4.1 Creating a servlet that uses JDBC to access Domino R5 data	76
4.4.2 Creating a JSP that uses JDBC to access Domino R5 data	82
4.5 Accessing Domino R5 using the Domino R5 Java classes	87
4.5.1 Creating a servlet that sends a Domino e-mail	87
4.5.2 Accessing a remote server from a servlet using IIOP	92
4.5.3 Accessing a Domino R5 server from a JSP	94
4.6 Accessing Domino from EJBs	96
4.6.1 Writing an EJB which uses the Domino Java API	97
4.6.2 Deploying the EJB	100
4.6.3 Redeploying the EJB	102
4.6.4 Summary	103
Chapter 5. Using WebSphere from Domino R5	105
5.1 Invoking servlets from Domino R5	105
5.1.1 Servlet URLs	105
5.1.2 Passing data to servlets in the URL	108
5.1.3 Posting data to servlets from Domino R5 forms	111
5.2 Calling JavaServer Pages from Domino R5	115
5.2.1 JavaServer Page URLs	115
5.2.2 Passing data to JavaServer Pages	116
5.3 Calling Enterprise Java Beans that are managed by WebSphere	118
5.3.1 Prerequisites for calling WebSphere EJBs using RMI	119
5.3.2 Getting the client stub via the naming service	120
5.3.3 Creating an EJB and calling the methods it provides	122
5.4 Using Enterprise Java Beans from a Domino R5 Java agent	122
5.4.1 Invoking EJBs from Domino R5 Java agents directly	123

5.4.2 Using an RMI server to access EJBs	134
5.4.3 Invoking an EJB from a Domino R5 Java agent via a servlet . .	141
5.4.4 Summary	144
Chapter 6. Authentication and authorization	145
6.1 Approaches for directory sharing	146
6.1.1 Sharing the Domino Directory	146
6.1.2 Sharing another LDAP directory	146
6.2 Using the Domino Directory	147
6.2.1 Configuring the Domino Directory for LDAP access	147
6.2.2 Configuring WebSphere to use the Domino Directory	150
6.3 Using another LDAP directory	155
6.3.1 Installing and configuring the IBM SecureWay Directory	155
6.3.2 Configuring WebSphere to use the SecureWay directory	157
6.3.3 Configuring Domino to use the SecureWay directory	162
6.4 Using basic authentication in Domino and WebSphere	165
6.4.1 Configuring Domino for basic authentication	165
6.4.2 Configuring WebSphere for basic authentication	167
6.5 Protecting a Web application under Domino and WebSphere	169
6.5.1 Protecting Domino components of a Web application	170
6.5.2 Protecting WebSphere components of a Web application	171
6.5.3 Using basic authentication in our enterprise application	176
6.6 Summary	177
Chapter 7. Scalability and redundancy with Domino and Websphere	179
7.1 WebSphere Performance Pack	179
7.1.1 Network Dispatcher	180
7.1.2 Web Traffic Express	181
7.1.3 AFS enterprise file system	184
7.2 Domino Internet Cluster Manager	185
7.3 Domino authentication and WebSphere Performance Pack	187
7.3.1 Domino authentication reviewed	187
7.3.2 Network Dispatcher and Domino	191
7.3.3 Web Traffic Express and Domino	193
7.4 Caching implications of Domino Views and URLs	198
7.5 Recommendations	201
7.6 A note about tools and techniques	202
7.7 Summary	203
Appendix A. Using the IIS Web Server for Domino and WebSphere .	205
A.1 Installation of IIS 4.0	205
A.1.1 Configuration of Internet Information Server	207

Appendix B. WebSphere plug-in behavior and tracing options	217
B.1 WebSphere HTTP server plug-in behavior.	217
B.2 Tracing Domino and WebSphere	220
B.2.1 Tracing the Domino LDAP task	220
B.2.2 Tracing in WebSphere	225
Appendix C. Additional Web material	231
C.1 How to get the Web material	231
Appendix D. Special notices	233
Appendix E. Related publications	237
E.1 IBM Redbooks	237
E.2 IBM Redbooks collections.	238
E.3 Other resources	238
E.4 Referenced Web sites.	239
How to get IBM Redbooks	241
IBM Redbooks fax order form	242
Index	243
IBM Redbooks review	247

Preface

This redbook is about using Lotus Domino R5 together with IBM WebSphere 3.0.2. We describe how to install Domino and WebSphere on Windows NT, and then look at how the components in a WebSphere Web application (servlets, JavaServer Pages and Enterprise JavaBeans) can work together with the services offered by the Domino Object Model. Our examples, developed in VisualAge for Java and Domino Designer are available for download from the IBM Redbooks Web site.

We also show how to use Domino Directory or IBM SecureWay as a shared LDAP directory for a combined Domino and WebSphere application, and we describe the necessary steps to protect all elements of the application. Finally we discuss options for scalability and redundancy when using Domino with its Internet Cluster Manager or the WebSphere Performance pack.

This redbook enables architects and developers at business partners, solution developers and customers to understand how Domino and WebSphere integrates from a technical angle.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Cambridge Center.

Søren Peter Nielsen works for the International Technical Support Organization at Lotus Development, Cambridge, Massachusetts. He manages projects that produce redbooks about all areas of Lotus products. Before joining the ITSO in 1998, Søren worked as an IT Architect for IBM Global Services in Denmark, designing solutions for a wide range of industries. Søren is a Certified Lotus Professional at the Principal level in Application Development and System Administration.

Mike Bartlett is an IT Architect with IBM Global Services in Toronto, Ontario, Canada. As an architect with the Domino-Notes Practice, his main role is the development of strategic e-business solutions for IBM customers, including corporate extranets, messaging solutions and custom application design. Mike has over 26 years experience in consulting with client organizations in the insurance, distribution, finance, telecommunications, retail, and manufacturing industries.

Michael Laskey is an IT Architect with IBM Global Services in Tampa, Florida, USA. As a Lead Developer in the e-Commerce Development and Support organization, his main role is the development of service and custom solutions for IBM customers, especially those related to collaboration, messaging and business-to-business applications. Mike has over 20 years experience with software development, including operating system, client-server, networking and messaging systems.

Christian Steege is a Systems Architect at Lotus Professional Services in Zürich, Switzerland. He graduated in Information Management at the University of St. Gallen, Switzerland. Since 1990 he has developed Notes/Domino applications, first at the University of St. Gallen and then at different Lotus Business Partners. In 1998 he joined the Lotus Professional Services Team in Zürich. He leads the implementation of Domino solutions for Lotus customers in Switzerland.

Adrian Walmsley is a Consultant IT Architect in the IBM UK Software Business, based in Hursley, England. He has over 30 years of experience in the IT industry. His most recent interests include Domino/WebSphere integration, and in 1999 he pioneered workshops on this subject for IBM and Lotus personnel in Europe. As an assignee to Poughkeepsie in the mid-1970s, he was involved in the inception of the ITSO redbook program, and since then he has contributed to a number of redbooks.

A number of people have provided support and guidance. In particular, we would like to thank **Arthur Fontaine**, Marketing Manager for Domino/WebSphere Integration at Lotus, for direction and contributions; as well as **Richard Werbin**, Vice President, Prudential Insurance for coming to Cambridge to give us feedback.

In addition, we would like to thank the following people:

- Charlie Brown, Lotus
- Kurt Deitrick, IBM
- Stephen Londergan, Lotus
- Aimee Stone Munsell, IBM
- Nataraj Nagaratnam, IBM
- Patrick Xuereb, Lotus
- Alison Chandler, ITSO Poughkeepsie
- The ITSO Poughkeepsie editing team

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 247 to the fax number shown on the form.
- Use the online evaluation form found at ibm.com/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

X Domino and WebSphere Together

Chapter 1. The platform for collaborative commerce

If you're reading this Redbook you're probably already familiar with the power of Lotus Domino or IBM WebSphere. You may know Domino as an unrivaled tool set and environment for collaborative applications such as discussions, teamrooms, knowledge bases, sales force automation, etc. You may appreciate the additional capabilities of Lotus Framework products such as Sametime for instant messages and meetings, QuickPlace for Web-based projects and teams, Domino.Doc for document management, and Domino Workflow for development of advanced process automation applications. Or, you may already be using the IBM WebSphere software platform to rapidly develop and deploy dynamic web sites, e-commerce stores or B2B marketplaces.

1.1 The opportunity: Collaborative commerce

IBM WebSphere, Lotus Domino, and their related products can be used to deliver new value to customers in the form of *collaborative commerce* services. Collaborative commerce is a term used by industry experts such as the Gartner Group (*C-Commerce: The New Arena for Business Applications*. Gartner Group, August 1999) to describe the next generation of e-business applications that enable organizations to “*more quickly find new, different and innovative ways of solving business problems and capturing new business.*” Gartner predicts that this will be the “dominant application model by 2004.”

Collaborative commerce moves beyond static content sites and current-generation transactive sites, to create a model that maps much more closely to the way traditional commerce is conducted. Consider how brick-and-mortar companies capture and retain business. They offer the best customer care, create preferred supplier relationships, bring new products to market faster than competitors or offer lower prices. Certainly, information technology is critical, and companies must leverage constantly evolving attributes of capability, speed, and reliability. But, it is how they use technology to improve their internal processes and build better relationships with customers, partners, and suppliers that produces true competitive advantage. Even in the Internet era, businesses need to differentiate with customer care and communication.

1.2 The problem: Collaboration on a global scale

If all this is so obvious, why is collaborative commerce only now becoming a focus? It's really just a matter of solving the pain that hurts most, first.

Because there has never been a network with the scale and ubiquity of the Internet, there was no body of experience in supporting the massive amount of traffic and the huge volume of transactions that became possible with the explosion of the World Wide Web. Revenue impact and media coverage resulting from Web site failures, particularly during surge periods, put pressure on software vendors to build a new class of Web application servers.

As a result, today's largest e-business applications—merchants, brokerages, news and content sites—are built upon a new generation of Java-based Web application platforms such as the IBM WebSphere software platform. The design principle of these software servers is massive scalability and industrial-strength transaction support. It's the “big iron” mentality brought to the Web. Here, IBM's long experience with mission-critical applications, combined with new technologies such as Java Server Pages (JSPs), Java Servlets, and Enterprise Java Beans (EJBs), have propelled WebSphere application servers into a leading position in this white-hot market.

Pure Java is an excellent choice because it allows a virtually unlimited scope of capabilities and customization. The application logic is customized for every implementation, in the form of business objects such as servlets and EJBs. There are many prebuilt EJBs and servlets from which to start creating the type of transactional and integrated applications that make large scale e-business possible.

However, there are some things for which servlets and EJBs are not as well suited. You probably wouldn't write a threaded discussion, instant messaging or instant meetings utility, or document-based workflow process as a Java program. Domino does these things very well, and it's the combination of highly collaborative elements and highly transactive elements in the same application that offers the exciting possibilities.

WebSphere and Domino are poised to leverage each other as the only complete and integrated platform for collaborative commerce. Massively scalable and transactive applications will be augmented by new services focused on creating and nurturing relationships up and down the value chain.

1.3 The technology plan

While the benefits of collaborative commerce are compelling, the technology situation is a bit more complicated. As noted, the combined capabilities of Domino and WebSphere are unrivaled by any other product or vendor. What is changing over time is the ease with which the two application servers can be used in a single application. This redbook represents very much a

point-in-time discussion, and it will require updates to reflect the technical integration built into future versions of the Lotus Domino and IBM WebSphere application server products.

This redbook is written for the point in time of summer, 2000. Integration techniques are centered around common Java capabilities, APIs, and the broad open standards support of Domino and WebSphere. Later in 2000, some very important technical work will improve the integration of the two products, greatly easing the process of creating collaborative commerce application. Some of the things you can expect are:

- Common cookie

The ability to support a common cookie for shared authentication between Domino and WebSphere servers in the same domain will greatly simplify the development and administration of applications. For example, users will be able to traverse all points of the application without the need to log in multiple times, and shared sessions will allow for information and context to be easily passed between Domino and WebSphere.

- LotusScript in JavaServer Pages

A `<Script Language="LotusScript">` option will enable BASIC programmatic access to the full range of Domino classes and methods.

Over the longer term, plans call for ever greater integration of Domino and WebSphere. Some of the areas being considered are:

- Common installation and administration

A single user interface for setup, configuration and management.

- LotusScript as pervasive programming language

LotusScript will become the BASIC-compatible development model in WebSphere and WebSphere family products such as WebSphere Commerce Suite and B2B Integrator.

- Tools integration

Research underway will determine ways to make it easier for Java and Domino developers to manipulate a full range of Domino and WebSphere objects from within their preferred integrated development environment (IDE).

- Platform Integration

With a common foundation between core Domino and WebSphere servers, it will be easy to integrate framework products such as WebSphere Commerce Suite or B2B Integrator, Lotus Sametime or QuickPlace.

1.4 Collaborative commerce with Domino and WebSphere

Let's look at the types of real-world applications that benefit from the collaborative commerce model. Lots of applications have a need for both structured and/or high volume data, as well as unstructured or process-centric data. Coming at that with any single product—even WebSphere or Domino—is inherently suboptimal somewhere. This is nothing more than the age-old adage, "The right tool for the job."

For example, Domino excels at distributed authoring, document management and customer support, and provides strong connectivity to a variety of RDBMS, ERP and transaction systems. With its 64 Gb database limit and optimized document store, Domino will perform quite well for all but the highest-volume applications. But for applications that must handle large quantities of relational data, massive or uncertain traffic levels, real-time data presentation or high-volume transactions against multiple sources, WebSphere is a more natural choice.

WebSphere offers industry-leading support for the J2EE enterprise Java specification, and is a card-carrying member of the IBM family of e-business products—the brand that industries and governments around the world turn to when it's got to work right, all the time, no excuses allowed.

In this context, applications naturally partition themselves. Applications like supply chain management (SCM), customer relationship management (CRM), or, increasingly, e-commerce, require best-of-breed application services across boundaries between structured and unstructured content, between transaction-oriented processing and collaborative decision making.

1.5 The IBM WebSphere e-business platform

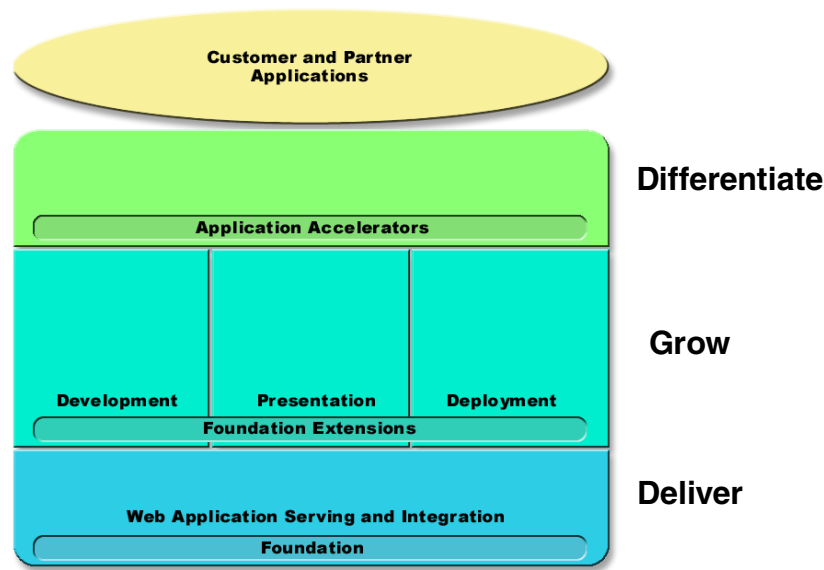


Figure 1. The IBM WebSphere e-business platform

This diagram illustrates the IBM e-business foundation. Based on the thought that varied services will be required in next-generation applications, teams from throughout IBM worked to create a platform, or foundation, under which the different products could operate. Essentially, it takes the common or duplicated services among the products—things like security, programming models, and Java support that are required by any viable application server offering—and makes them common across all products (Foundation). It augments the common services with tools to create and manage applications (Foundation Extensions). It places the unique services at a top, accessible layer where they are acquired as particular brands. And, of course, the entire model is built on the assumption that consultants, solution developers, integrators, and corporate developers will apply the toolset against specific, real-world problems, as has always been the case.

There are two immediate benefits. Each product now has access to best-in-class services of the types that they may not have been optimized for in their core designs. For example, WebSphere was not designed for workflow, but can gain it via Domino. At the same time, it means that customers can acquire any IBM e-business software with full confidence that a set of proven complementary capabilities can be added easily at any time.

Associating products and brands with the platform, this diagram shows that the foundation layer is largely comprised of IBM WebSphere Application Servers, with capabilities from Domino such as directory and script support. The foundation extensions layer includes all the development and administration tools that allow you to extend and grow an established Web application. The application accelerator layer of the platform is where the specific business value resides, in the form of the products and brands that address specific needs: Domino, WebSphere Commerce Suite, IBM B2B Integrator, and so on.

The top oval is important because it addresses the continued role of partners, solution developers and corporate developers. An application server is still judged by the applications created with it. We've separated foundation services from application services in the architecture, but are in no way moving up into the solutions space with this strategy. In fact, IBM and Lotus are trying to provide an integrated family of products that will let developers focus more of their time on rapidly developing and deploying packaging and custom solutions that help customers get on the web, rapidly grow and adapt to meet new market requirements, and stay ahead of the competition.

1.6 Summary

We have now given you the big overview of how Domino and WebSphere fit together, now and in the future. In the rest of the chapters in this book we will delve into the technical details of how they work together.

Chapter 2. Installation and setup

This chapter describes how to install and configure Domino and WebSphere to work together. There are many possible variations of a Domino and WebSphere installation. We do not intend to replace the installation guides included with the products. Instead, we will walk through a scenario where we install Domino Application Server R5 and WebSphere Application Server Advanced Edition V3.02.1 on the same computer running the NT4 Server operating system. In addition, we also mention how you can install and configure Microsoft IIS to act as HTTP server for the Domino and WebSphere setup.

The terms *WebSphere* and *WebSphere Application Server* (or WAS) will be used interchangeably here, even though WebSphere refers to an entire product line. Other products will be referred to by their full names.

The intended audience is a Domino or WebSphere developer with some Domino administrator experience who wishes to test WebSphere-Domino integration.

2.1 Introduction

The high level view of the installation steps is:

1. Check for necessary prerequisites and network configuration.
2. Log on as a user with administration privileges for WebSphere and DB2.
3. Install DB2 UDB and its current fixpack.
4. Install the IBM Java Development Kit (JDK).
5. Install and configure Domino R5.
6. Install WebSphere V3.02, configure, apply service upgrade.
7. Reconfigure Domino R5 to use the WebSphere Domino 5 plug-in.
8. Verify servlets can be loaded successfully.

2.2 Prerequisites

In this section we list the hardware and software requirements for the machine you want to install on, as well as the different product software levels required.

2.2.1 Platform

Hardware:

- Pentium II or higher, 256 MB absolute minimum, 512 MB recommended. We used 256MB for our testing with satisfactory results.
- At least 500 MB free on the drive or drives used to install the products. The disk space requirements of the products after installation are:

DB2	180 MB
JDK	30 MB
Domino	500 MB
WebSphere	120 MB

Note that the space requirement for Domino can be reduced to a certain extent by choosing to install fewer components. For example, it is possible to not install the help files. These files alone require 140 MB.

Software:

- Microsoft Windows NT v4.0 SP4 or SP5. Either workstation or server code can be used.
- TCP/IP networking with a fixed IP address (or a loopback adapter for a standalone machine).

2.2.2 Product software levels

The product software levels we used were:

- DB2 UDB 6.1 Enterprise Edition plus fixpack 4.
- IBM JDK 1.1.7p Build Level 0823.
- Domino R5.0.4.
- WebSphere Application Server V3.02.1 (and PTF PQ37562 if IIS support is needed).
- Microsoft Internet Information Server (IIS) 4.0 installed on NT Server 4.0 (Service Pack 5). Using IIS is optional.

Higher versions of the products should also work. The highest level currently available product should be used except where specifically stated otherwise.

2.3 Creating a user with administration rights for DB2 and WebSphere

WebSphere Application Server, Domino, and DB2 must run under the permissions of a user or as system services. For testing purposes, it is more flexible to use a user ID with rights to run as an extension of the operating

system rather than load the products as system services (however, WebSphere Application Server and DB2 *must* run as system services). For a production system, these products should run as services so that they will automatically load when the system is started without operator signon.

In this section we describe how to create an NT user ID with rights to run as an extension of the operating system. To do this, you must have the right to create an ID on your local machine (and in any NT Domain Control server's user registry if your machine logs on to a network).

User ID must not be longer than 8 characters

The user ID must be 8 characters or less to work with DB2. Thus, the default NT Administrator user name *Administrator* will not work with DB2. We used *db2admin* for our testing.

If you already have a user ID no longer than 8 characters and with the right permissions, you can skip this section.

A new user ID is created using the NT User Manager.

1. Start the NT User Manager by selecting:

Start -> Programs -> Administrative Tools (Common) -> User Manager

2. In the User Manager select:

User -> New User

This brings up the New User dialog.

3. Fill out the user information as shown in Figure 2 on page 10.

Make sure to specify a password you can remember.

Change the default setting for password expiration so only *Password Never Expires* is selected

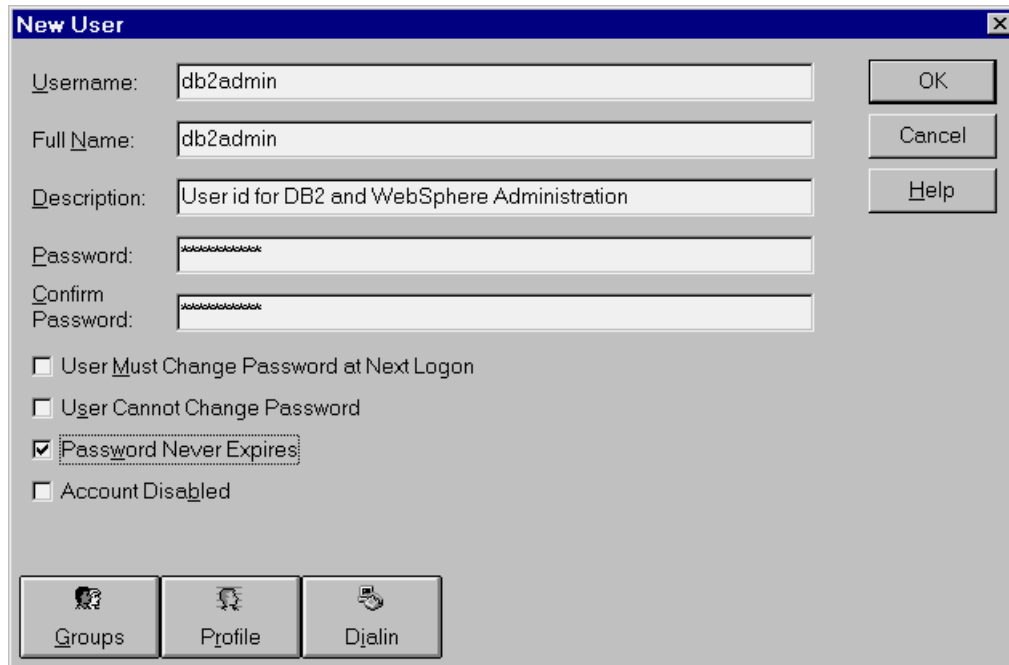


Figure 2. Adding a new user to run DB2, WebSphere and Domino

4. Click the **Groups** button to add our new user to the Administrators group. You do this by selecting the **Administrators** group on the right pane and clicking the **<-Add** button as shown in Figure 3 on page 11.

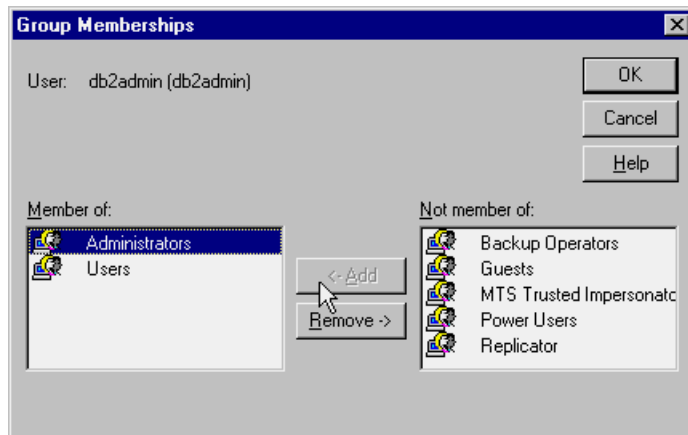


Figure 3. Adding the new user to the Administrator's group

Click OK twice to return to the NT User Manager main window.

5. Select **Policies -> User Rights** from the menu in NT User Manager to set the user just created to run as part of the operating system.

- Add the newly created user to the list we can assign rights to.

Click the **Add** button. A dialog box named Add Users and Groups appears.

Click the **Show Users** button. Select our user ID (db2admin) in the listbox and click the **Add** button. Return to the Users Rights dialog by clicking **OK**.

- Make sure our user (db2admin) is selected in the list. Select the option **Show Advanced User Rights** and then select **Act as part of the operating system** in the listbox where you specify rights, as shown in Figure 4 on page 12.

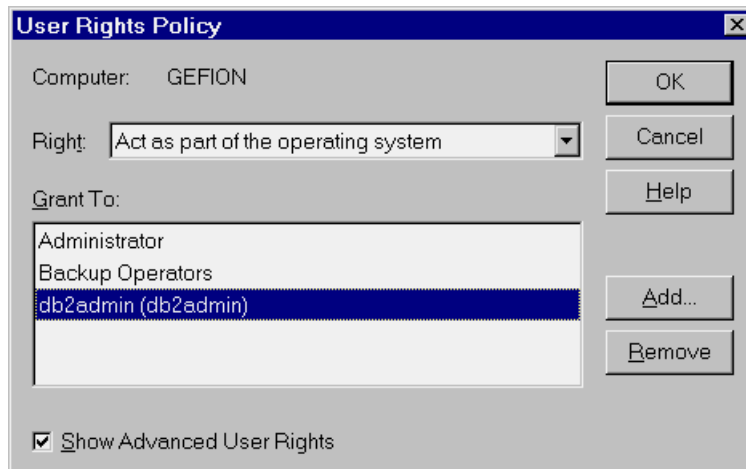


Figure 4. Setting the new user to run as part of the operating system

Click **OK** to confirm the changes and close the dialog box.

6. Finally, close the User Manager.

Restart NT before you log on using the new user ID (db2admin) to make sure that the system rights for db2admin are in effect system wide. If you are unable to log on with this ID, you need to resolve the situation (perhaps with your NT administrator's help) before proceeding. It may be that you will want to create a stand-alone NT server not associated with your existing NT Domain for testing purposes.

2.4 Installation of DB2 UDB Release 6.1 and fixpack

WebSphere requires a database system that supports Java to store its configuration and state information in the WAS database. If your system does not have DB2 or one of the other database systems supported by WebSphere, the Websphere installation will automatically install DB2 Universal Database (UDB), but it is then limited to only development purposes. For a production environment, or to have access to all of the DB2 functionality, you should obtain the DB2 package separately.

The preferred supported level of DB2 is 6.1 plus fixpack 2 or higher. We used this level, but other levels are supported according to the WebSphere product documentation.

Make sure you are logged on to Windows NT with the user ID you created in 2.3, "Creating a user with administration rights for DB2 and WebSphere" on

page 8, or another user ID with similar rights. Although this is not strictly necessary to run the install program, you will need to be logged on with this user ID when accessing DB2 after installation.

2.4.0.1 Installation of DB2 UDB Release 6.1

DB2 UDB can be installed either from a zip file from the IBM Software Web site or from a product CD. If the installation program on the CD doesn't start automatically (or if you are installing from an unpacked zip file) you must run the setup.exe program manually.

1. First, the welcome screen is shown. Click **Next**.
2. Then the selection screen shown in Figure 5 is displayed.

Accept the default “DB2 WorkGroup Edition” or “DB2 Enterprise Edition” (the message displayed will depend on the edition of DB2 purchased).

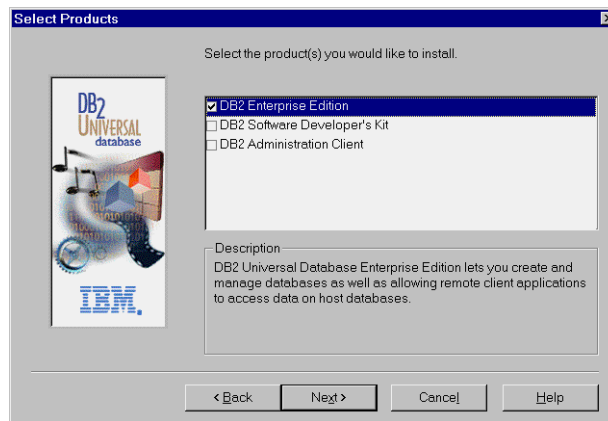


Figure 5. Product installation options

We also recommend that you select “DB2 Software Developer’s Kit” if you want to develop applications that use DB2.

Click **Next** to continue.

3. You have to select between a *Typical*, *Compact*, or *Custom* installation.

Click **Typical**.

4. On the next screen you can select where to install DB2.

The installation directory can be changed to a disk with adequate space if the default drive (C:) does not have enough space. Click the **Browse** button as shown in Figure 6 on page 14 to specify a new location for DB2.

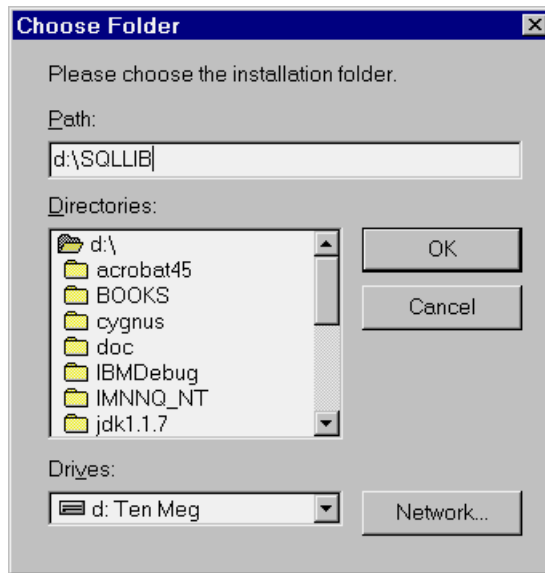


Figure 6. Selection of installation directory

In our example we used:

D:\SQLLIB

Click **OK** to return to the installation screen and **Next** to continue.

If you get a message indicating the directory you selected does not exist, accept to have it created by clicking **Yes**.

5. You are then prompted to supply a user ID and password for the DB2 administration server to run under.

Replace the defaults provided with the user ID set up earlier with NT user administration. This user ID will also become the default user ID in DB2.

Note: The installation program suggests the user ID db2admin with the password db2admin. You have to make sure you overwrite the password fields with the password you have chosen.

Click **Next** to continue.

Note: If you didn't restart NT after you created your user you will get a message about the installation program not being able to validate the user ID. In this case you can choose to continue anyway.

6. The next screen displays the installation options chosen. Click **Next** to confirm your choices and start the installation of the program files.

7. When the files are installed you are prompted to reboot your computer to complete the installation.

Click **OK** to reboot the computer.

2.4.0.2 Confirmation of successful installation

If your installation of DB2 was successful, the First Steps screen shown in Figure 7 will be displayed after you have rebooted your computer.

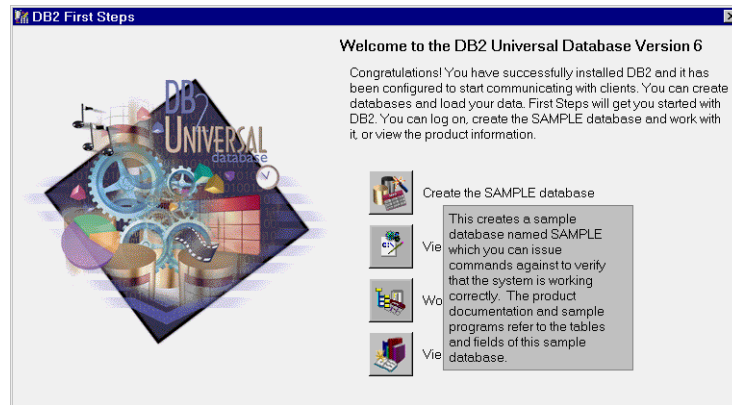


Figure 7. DB2 First Steps successful installation confirmation screen

The top button on this screen will create a sample database to allow you to test that DB2 UDB is working correctly.

- Select to create the sample database by clicking the button.
- A warning message saying this operation might take several minutes is displayed. Select **Yes** to continue creating the sample database.
- A message window saying *Create Sample database* is shown while the database is being created and once completed a message box informs you that it was created successfully. Click **OK** to dismiss the message box.
- To view the content in the sample database click the button labeled **View the SAMPLE database** on the First Steps panel.

This will start the *DB2 Command Center* and prompt you for a user ID and password. Enter the user ID and password you specified during the DB2 installation (db2admin) and click **OK**.

The command center opens with a predefined script as shown in Figure 8 on page 16.

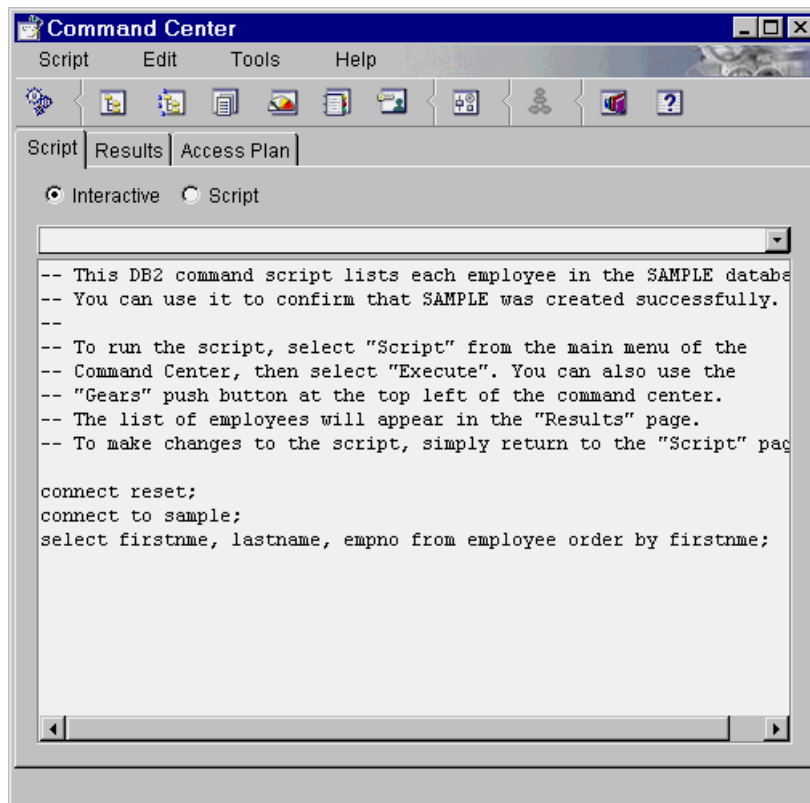
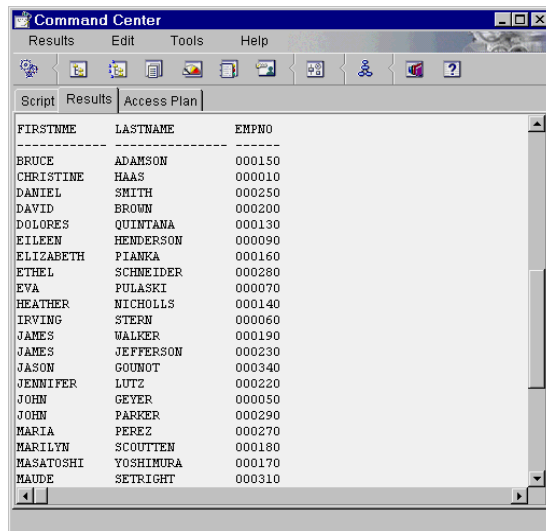


Figure 8. DB2 Command Center testing the sample database

Click the **Gearwheel** icon on the left of the button bar to execute the query shown in the script window. The results will be shown in the Results tab, as shown in Figure 9 on page 17. You may need to scroll the display to see the returned list of names.



The screenshot shows a window titled "Command Center" with a menu bar (Results, Edit, Tools, Help) and a toolbar. Below the toolbar are three tabs: "Script", "Results", and "Access Plan". The "Results" tab is active, displaying a table with three columns: "FIRSTNAME", "LASTNAME", and "EMPNO". The table contains 20 rows of employee data.

FIRSTNAME	LASTNAME	EMPNO
BRUCE	ADAMSON	000150
CHRISTINE	HAAS	000010
DANIEL	SMITH	000250
DAVID	BROWN	000200
DOLORES	QUINTANA	000130
EILEEN	HENDERSON	000090
ELIZABETH	PIANKA	000160
ETHEL	SCHNEIDER	000280
EVA	PULASKI	000070
HEATHER	NICHOLLS	000140
IRVING	STERN	000060
JAMES	WALKER	000190
JAMES	JEFFERSON	000230
JASON	GOUNOT	000340
JENNIFER	LUTZ	000220
JOHN	GEYER	000050
JOHN	PARKER	000290
MARIA	PEREZ	000270
MARILYN	SCOUTTEN	000180
MASATOSHI	YOSHIMURA	000170
MAUDE	SETRIGHT	000310

Figure 9. Result set from sample database confirming successful configuration

Once the sample database has been tested, the fixpack for DB2 can be installed as described in the following section.

Close the Command Center. Accept to discard all changes when prompted.

2.4.1 Installation of DB2 fixpack

WebSphere V3.02.1 requires DB2 6.1 fixpack 2 or higher. You can download the current DB2 fixpacks from the support area of the DB2 Web site at:

<http://ibm.com/db2>

Look for *Maintenance for DB2 Universal Database Version 6.1 products* and select the fixpack that applies to your product. In our case we installed fixpack 4.

Before you install the fixpack you must stop all DB2 services.

1. Open the Windows NT Services Panel by selecting:

Start -> Settings -> Control Panel.

This opens the Control Panel folder.

2. Double-click the **Services** icon (a pair of gear wheels).

The Windows NT Services panel opens as shown in Figure 10 on page 18.

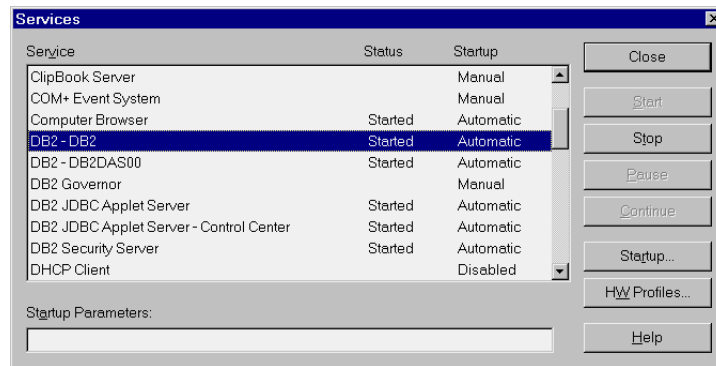


Figure 10. Stopping DB2 services in Windows NT Service panel

3. Stop all services in the panel beginning with DB2 by selecting them individually and clicking the **Stop** button.

There may be other services that use DB2 (such as Netfinity Support Manager) which should also be stopped. If any are missed the initial prompt on starting the fixpack installation will identify running services and give the option of stopping them before proceeding or cancelling the installation.

4. Once the services have been stopped, expand the fixpack zip file to a temporary directory and run the Setup.exe program.

The installation is similar in appearance to the original installation. The difference is that any selections already made in the original product installation will already be selected. For example, if the original installation was to D:\SQLLIB the fixpack will be installed to this directory.

5. You will be prompted to reboot the computer. Do this and the installation of the DB2 fixpack is complete.

2.5 Installing IBM JDK 1.1.7P

WebSphere V3.02.1 on NT requires the IBM Enhanced Java Development Toolkit (JDK) 1.1.7 or 1.1.8.

We used the IBM JDK 1.1.7, which we downloaded from the IBM Software Web site at:

<http://www.ibm.com/java/jdk/download/index.html>

The installation file was a self-extracting zip file named `ibm-jdk-n117p-win32-x86.exe`.

Note: The file name may have changed when you download your version of the JDK.

1. Run the file `ibm-jdk-n117p-win32-x86.exe`.

A message box will ask you to confirm that you want to install the JDK.

2. Click **Yes**. The installation program is unpacked and started. A welcome panel is shown.

3. Click **Next**, read the license agreement, and accept it by clicking **Yes**.

You will then see the panel shown in Figure 11.

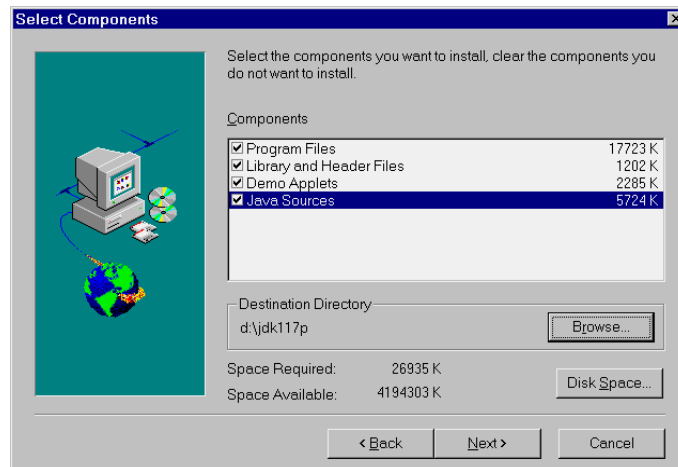


Figure 11. Installation options for IBM JDK 1.1.7P

4. In our case we changed the destination directory to the D drive, but you can leave the default selections as they are if you wish.

Click **Next** twice and the installation of the files will begin.

5. Click **Finish** to close the installation program once the files are copied.

You confirm that the installation was successful by doing the following:

- Open a command prompt.
- Change to the JDK installation directory, like this:

```
C:\>cd:  
D:\>cd jdk117p\bin
```

- Type:

```
java -fullversion
```

as shown in Figure 12 on page 20.

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text: C:\>d:; D:\>cd jdk117p\bin; D:\jdk117p\bin>java -fullversion; java full version "JDK 1.1.7 IBM build n117p-19990823 <JIT enabled: ibmjitc>"; D:\jdk117p\bin>_

Figure 12. Testing JDK installation version

You should get the following response:

```
java full version "JDK 1.1.7 IBM build n117p-19990823 (JIT enabled: ibmjitc)"
```

If you do not get this response you are either not in the correct directory or the JDK has not been installed correctly.

This test can also be done *before* installation, to check whether it is necessary. Note that it is possible to install WebSphere 3.02 with an earlier build of the JDK1.1.7 than the one shown, but this will make use of the WebSphere fixpack incompatible with the Domino plug-in described later.

2.5.1 Editing the system CLASSPATH to include the JDK

In order to avoid problems when running the WebSphere fixpack installation, ensure that the system CLASSPATH variable includes the path to the JDK1.1.7p Class.zip file. This file is located in the *lib* subdirectory under the directory you installed the JDK to.

Note that neither the JDK install (nor the subsequent WebSphere installation) will set the system CLASSPATH, so this will have to be done manually using the following steps:

1. Go to the Windows NT desktop and right-click the **My Computer** icon. Select **Properties** on the popup menu. This opens the System Properties window.
2. Select the Environment tab.
3. Highlight the CLASSPATH variable in the upper pane and edit it in the lower pane in the field named Value.

Include the absolute reference to the JDK Class.zip file *as the first entry* of the class path. We could not get the Websphere fixpack installation to run unless the reference was placed first in the CLASSPATH. In our case we added the following:

```
D:\jdk117p\lib\classes.zip;
```


4. Click **Set** to apply your changes. If you click OK or Apply without clicking Set first your changes will not be saved.

Your System Properties CLASSPATH should now look similar to Figure 13.

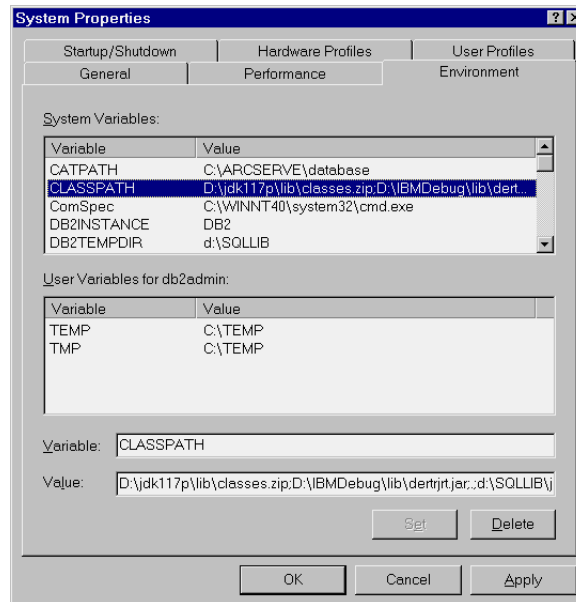


Figure 13. Setting the system CLASSPATH to point to the IBM JDK class.zip

5. Click **OK** to close the System Properties window.

Note: We changed the *system* setting for CLASSPATH. If there is a user variable CLASSPATH defined (in the lower pane on the Environment tab) you must either delete that variable so it doesn't overrule the system setting or you must add the JDK reference to that CLASSPATH as well.

2.6 Installing Domino Server R5

The Lotus Domino R5 Server family consists of Domino Mail Server, Domino Application Server and Domino Enterprise Server. If you want to do more with Domino than just use its HTTP stack you should install the application or the enterprise server.

To use the Domino HTTP stack WebSphere V3 requires Domino Server R5.0.2b or higher. We installed Domino Application Server R5.0.4 because prior versions had a problem that meant Domino and WebSphere couldn't communicate properly with each other via IIOP.

1. If you install Domino from a CD the installation program should start automatically. If it does not, start the installation by running **setup.exe** from the installation CD.

A welcome screen will be shown.

2. Click **Next**, read the license agreement and click **Yes** to accept it.
3. Enter your name and company on the next panel if the installation program hasn't been able to pick up this information from the system.

Click **Next**.

4. Specify where you want the Domino program and data files placed or accept the default locations.

Click **Next**.

5. Choose the type of server to install.

In our case we selected Domino Application Server as shown in Figure 14.



Figure 14. Selecting Domino Server type

You can use the **Customize** button to further refine the installation process. If you click **Customize** you will get a list of options to select or deselect as shown in Figure 15 on page 23.

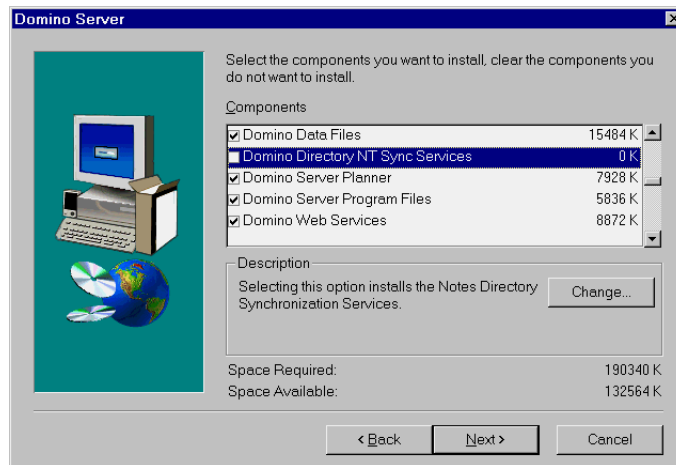


Figure 15. Selection of components to be installed

For example, you may not need the help files because you will install the help together with the administration client on another machine or in another directory later on. Deselecting help from the server install will save you around 140 MB.

If you change anything make sure that **Domino Web Services** is selected; other components can be selected or deselected at will. For testing purposes, you should not select the option to allow Domino to run as an NT service. It is possible to add more components later by running the installation program again and only selecting the desired components to be added.

6. Click **Next**. It does not matter whether you picked the default installation or the customized one. You will be brought to the panel where you select which Program Folder to add the Domino Server to.
7. Click **Next** to accept the default suggestion and to start the installation of the files.
8. Once all the files are installed the installation program shows a message saying that you can start the server by using the system menu (with the folder specified during installation) or by simply starting the server from the installation directory.

Click **Finish** to end the installation program.

2.6.1 Domino Server configuration and setup

After the Domino Server files have been installed the server must be configured before we can start it.

1. Start Domino by using **Start -> Programs -> Lotus Applications -> Lotus Domino Server** from the Windows NT task bar.

This will launch the Domino configuration program.

2. The initial configuration screen asks whether this is the first or an additional server in your Domino domain.

We selected **First Domino Server** to set up a stand-alone test domain as shown in Figure 16.

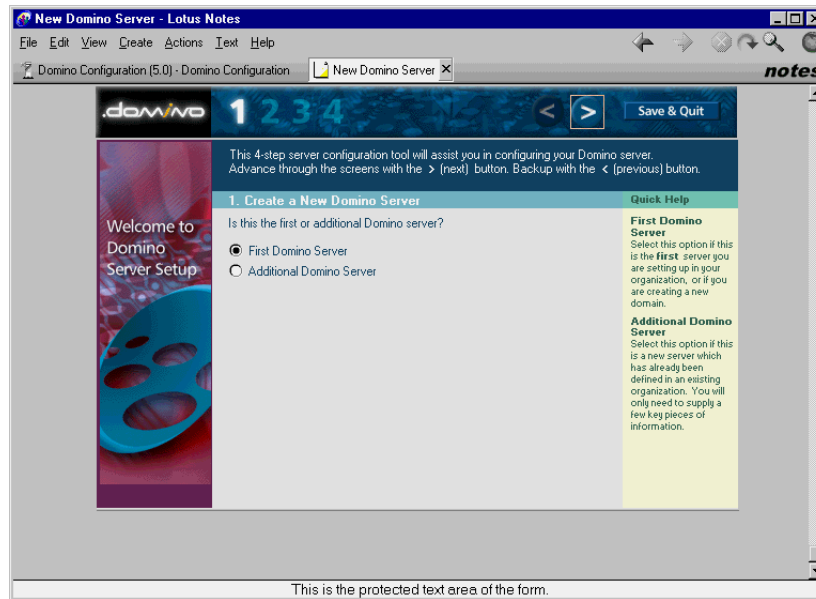


Figure 16. Initial Domino Server setup panel one

3. Click the forward button (>) at the top of the pane.

The next panel asks you to specify whether you want to set up the server using a quick and easy or an advanced configuration, as shown in Figure 17 on page 25.

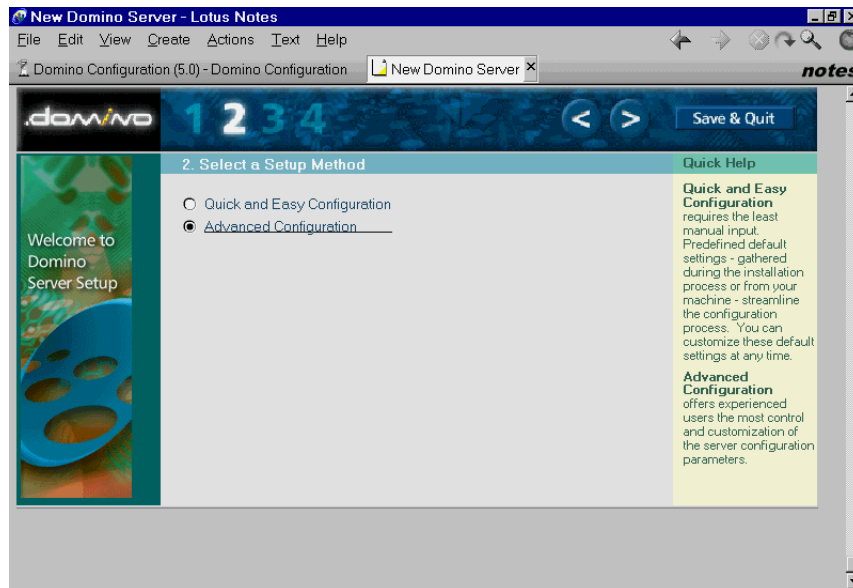


Figure 17. Initial Domino Server setup panel two

4. Select **Advanced Configuration** so that further options can be selected on the succeeding screen.

To continue, press the > button on the top bar.

5. The Advanced Configuration panel is shown.

We selected a reduced set of services for the server since we were only going to test it as a Web server. Note that the choices here only change the initial configuration setup of the server and can easily be changed later. For example, we specified using the Domino HTTP stack, but this is easily changed to use IIS during testing.

We added the following to the default set of services:

- HTTP for Web Browsers (Both Mail and Applications)
- POP3 for Internet Mail
- LDAP Directory Services

and we deselected the following additional services:

- Calendar Connector
- Schedule Manager

This is shown in Figure 18 on page 26.

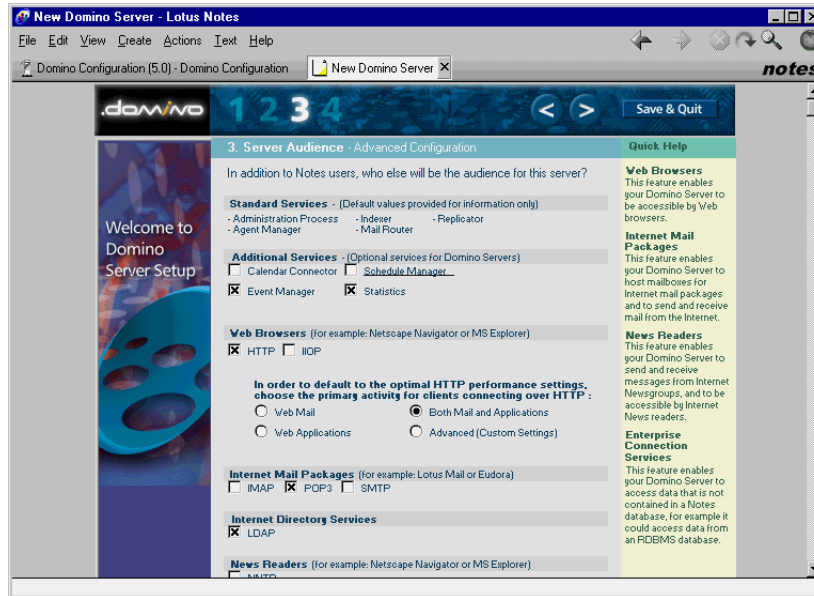


Figure 18. Initial Domino Server setup panel three

6. Once the desired initial services are selected, click the > button on the top to continue.

The Administration Settings panel will be shown.

- Here you specify the names of your Domino domain (*ITSO* in our case), certifier/organization name (*ITSO*), and administrator identity (*John Doe*).

Be sure to keep a record of the passwords you used for the certifier and the administrator since these will not be retrievable if you lose them.

- Select **Customize** in the Ports section under Network Options. You may have to scroll down a bit to do this. Then click the **Edit Ports** button. This opens a new window where you can see all communication ports that will be activated by default.

Disable all ports except **TCP/IP** and click **OK** to return to the setup panel.

The setup panel should now look similar to Figure 19 on page 27.

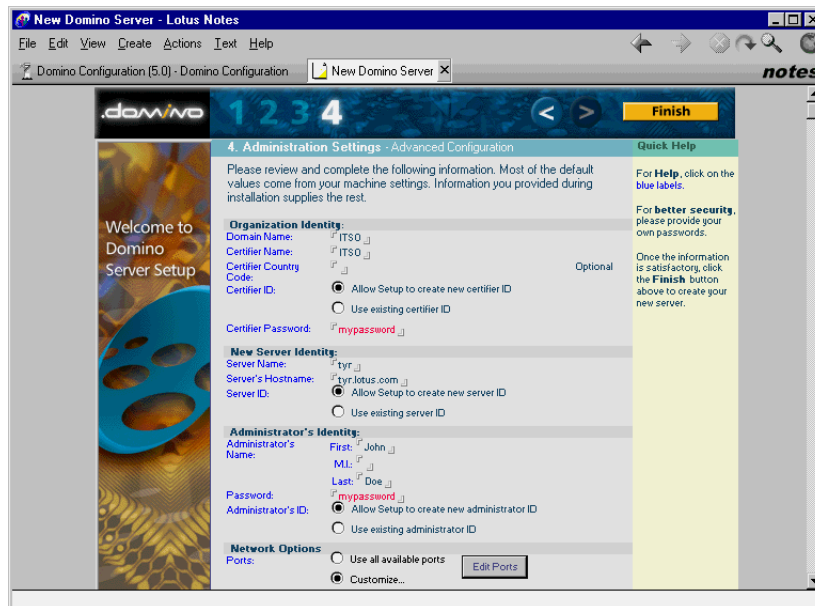


Figure 19. Initial Domino Server setup panel four

7. Verify all of the names and options are correct and click **Finish** to complete the server setup.

During the setup three Notes ID files will be created:

- Cert.id The certifier id file for the new organization.
- Server.id The server's id file.
- User.id The administrator's id file. By default, this will be saved in the administrator's person record in the Domino Directory.

Once the setup has finished you will see the panel shown in Figure 20 on page 28.

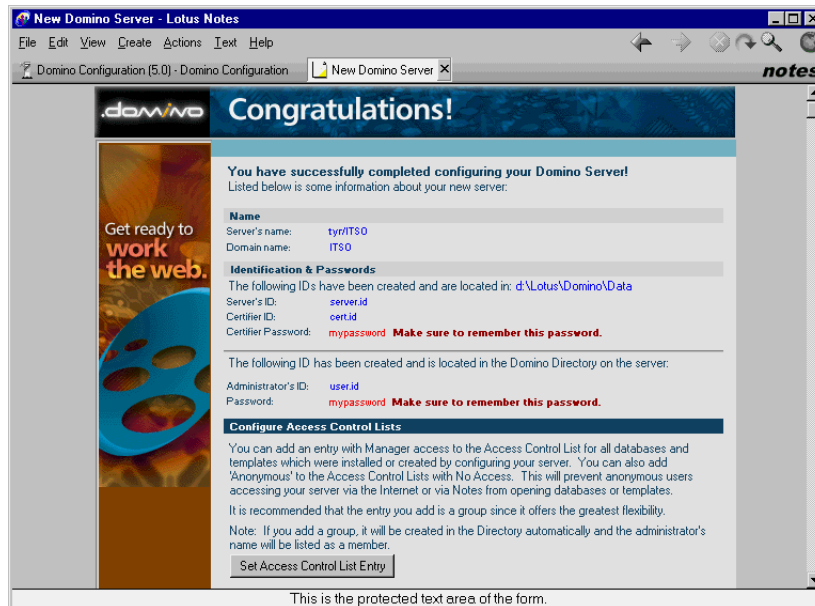


Figure 20. Final Domino configuration panel.

8. Click the **Exit Configuration** button to close the setup program. You may have to scroll down a bit to see the button.

2.6.1.1 Verifying the Domino server configuration

We will now check that the Domino HTTP stack loads correctly and it can be reached from a Web browser.

1. Start the Domino server using **Start -> Programs -> Lotus Applications -> Lotus Domino Server** from the Windows NT task bar.
2. Ensure that the HTTP task is running by issuing a `show tasks` command from the console or by observing the console log when the server starts.

Figure 21 on page 29 shows part of the output from the `show tasks` command. The top line confirms that the HTTP task is loaded.

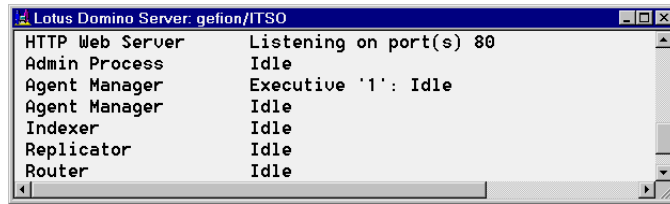


Figure 21. Domino Server Console showing that the HTTP task loaded

Note: The Domino HTTP task will fail to start if another HTTP task is running using the default HTTP port 80. For example, if you have IIS installed on your server, it should be stopped either from the Microsoft Management Console or from the NT Services panel.

3. Next, Web HTTP access to the Domino server can be checked by using a browser. Start your Web browser and type in the IP address of your Domino server as the URL, or if you are running on the same machine, you can just type `localhost` as the URL.
4. If Web browser access works correctly you will see the default R5 homepage in your Web browser, as shown in Figure 22.

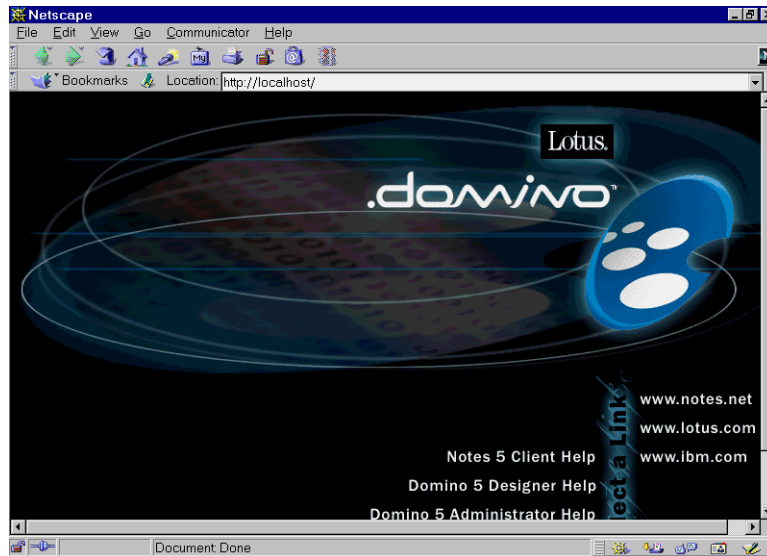


Figure 22. Home page displayed by Domino HTTP server

2.6.1.2 Adding the Domino program file to the path

You must add the directory with the Domino executable files to the system path. This is necessary since the WebSphere installation will later look for

some executables (*js32.dll* and *nlsctr.dll*). If the path statement is not updated you will get an error message when you attempt to start WebSphere and you have to uninstall WebSphere and try again.

The procedure for changing the system path is the same as we described in 2.5.1, “Editing the system CLASSPATH to include the JDK” on page 20, using the System Properties window.

In our case we added the following directory to the PATH

```
D:\Lotus\Domino;
```

Note: Remember to click **Set** on the Environment Variables tab before you close the System Properties window or your changes will be lost.

2.6.1.3 Installing the Domino Administration client

You must install the Domino Administration client on your server or another workstation (Lotus recommends using a separate workstation for administration). This will allow you to change the server’s settings easily, although much of the testing we describe can be managed by direct access to the server’s text console as illustrated in Figure 21 on page 29.

We will not describe the installation of Domino Administrator in detail. Be sure that the **Administration Client** is selected for installation; you can accept all other default values during the installation

This completes the installation and configuration of Domino R5 for the purposes of this exercise.

2.7 Installing WebSphere V3.02 and its service upgrade

We describe installing both WebSphere Application Server and its service upgrade (to raise it to V3.02.1); we performed our tests on WebSphere Application Server Advanced Edition V3.02.1.

WebSphere Application Server *Standard Edition* is included for free with Domino Server R5. The installation steps for WebSphere Standard Edition are slightly different than the ones we describe here. However, we installed the *Advanced Edition* because of its support for Enterprise Java Beans. You can download an evaluation version of the advanced edition from the following Web site:

<http://ibm.com/websphere>

2.7.1 Installing WebSphere Application Server V3.02

You can either install Websphere from a product CD or one large installation file. If you use a product CD the installation program should start automatically when you insert the CD. If not, you have to run the setup.exe program on the CD. If you have one big file, simply start the installation by running the file. Note that the installation process will require 70 MB or more free in the system TEMP directory (normally the C drive) even if installation is to another drive.

If your installation file supports installation using different languages, you must pick **English** to follow the procedure we describe below.

1. The first installation panel says your Web server must be shut down before proceeding with the installation.

Make sure Domino (and any other HTTP server) is shut down. Shut down the Domino server using the command `q` on the Domino server console.

Click **Next** to continue with the WebSphere installation.

2. The Install Options panel is displayed.

Modify the installation directory as necessary to select a drive with adequate space. In our example we used:

`D:\WebSphere\AppServer`

Select the radio button for **Custom Installation** and click **Next**.

3. Select the Web server you want to use together with WebSphere. WebSphere will install plug-ins for the Web server(s) you select.

Select **Lotus Domino V5.0 or higher**.

Later we will show how to use **Microsoft IIS V4.0** as the Web server for Domino and WebSphere so we also installed the plugin for IIS as shown in Figure 23 on page 32. If you don't want to work with IIS you can just pick Domino as your Web server. However, it will not hurt to select all the HTTP servers you wish at this stage—this selection will only copy the plug-ins to the WebSphere bin directory during file copying.

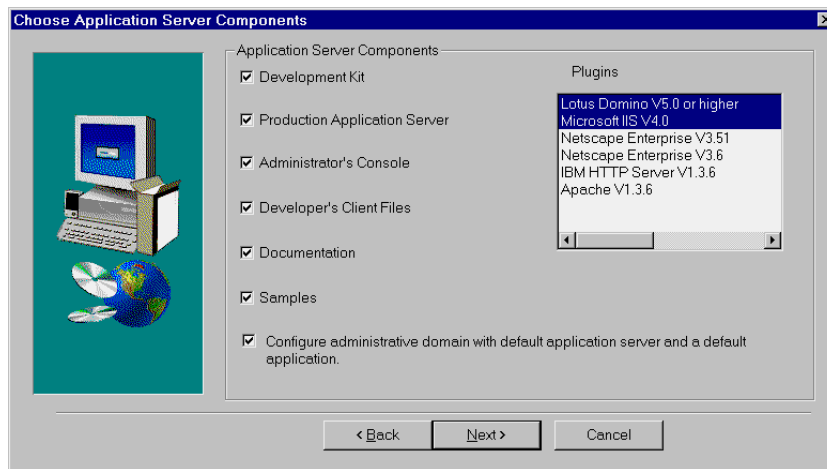


Figure 23. Selecting plug-ins and installation components for WebSphere 3.02

Also select the bottom check box option **Configure administrative domain with default** This is necessary to install the sample servlets, the servlet engine and an EJB container, which we will use later in this book.

Click **Next** after making these selections.

4. The installation program finds and displays the location of the IBM JDK as shown in Figure 24.

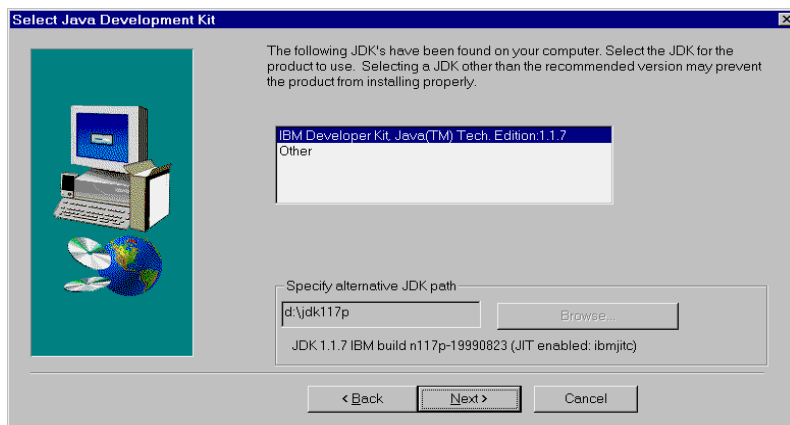


Figure 24. Confirming IBM Java Development Kit installation

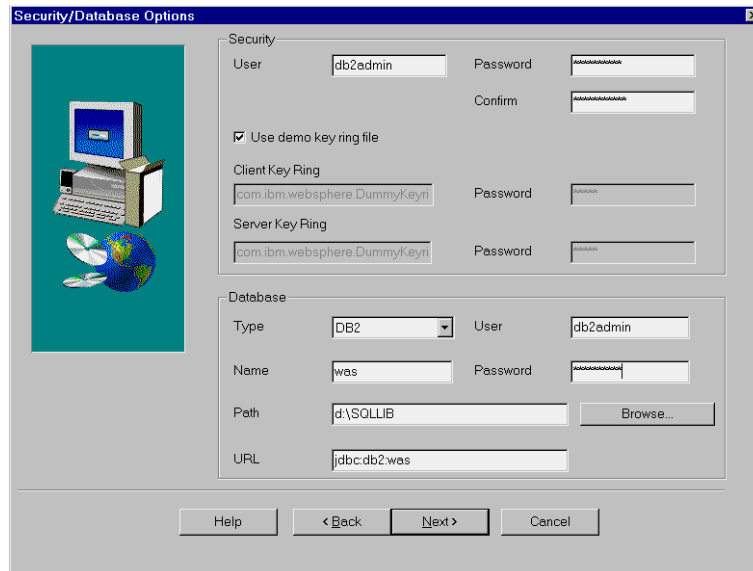
If there are other JDKs installed, select the IBM JDK1.1.7.

Note: If the Sun JDK 1.2 has been installed, it should be uninstalled before proceeding. This is because it will install itself to the Windows system directory. This causes it to take precedence over other JDKs such as IBM's. This is true even if the CLASSPATH variable is set to point directly to the IBM JDK.

Once the JDK version and location have been confirmed, click **Next**.

5. The next panel is the Security/Database Options panel. This configures the database access and user IDs to be used by WebSphere.

The database access is used by WebSphere to store the server configuration in a set of tables. Figure 25 shows the panel as we filled it in.



The screenshot shows the 'Security/Database Options' dialog box. On the left is a graphic of a computer monitor, keyboard, and CD-ROM. The 'Security' section contains fields for 'User' (db2admin), 'Password' (masked), and 'Confirm' (masked). There is a checked checkbox for 'Use demo key ring file'. Below are 'Client Key Ring' and 'Server Key Ring' sections, each with a text field containing '.com.ibm.websphere.DummyKeyri' and a 'Password' field (masked). The 'Database' section has a 'Type' dropdown set to 'DB2', a 'User' field (db2admin), a 'Name' field (was), a 'Path' field (d:\SQLLIB) with a 'Browse...' button, and a 'URL' field (jdbc:db2:was). At the bottom are buttons for 'Help', '< Back', 'Next >', and 'Cancel'.

Figure 25. Security and database setting for WebSphere Application Server installation

We set the user ID to **db2admin** (as previously described in 2.3, “Creating a user with administration rights for DB2 and WebSphere” on page 8) for both the WebSphere user and the DB2 user ID. All of the fields (other than the IDs and passwords) should already be filled in; if not, adjust them as shown in Figure 25.

Once this is done, click **Next**.

6. The next panel allows you to specify the program folder for WebSphere. Accept the default suggestion and click **Next** to start the installation of the program files.

7. Once the files are installed the final confirmation panel is displayed. Deselect the option to see the README file and click **Finish**.
8. You will be asked whether you want to reboot your computer now or later. Select to restart the machine now.

This is necessary to allow the configuration to be completed on the next system restart. Specifically, when the server is restarted the WebSphere configuration database (WAS) will be created in DB2.

Note: If you get a message during the restart of the computer saying something like *db2nq.exe... unable to locate DLL... js32.dll* then you haven't set the system path to point to the Domino executable files. In this case you have to uninstall WebSphere, set the path to the Domino files and install WebSphere again.

2.7.2 Installing the WebSphere 3.02 service upgrade

We need to apply a service upgrade to WebSphere to V3.02.1 to fix a problem in the Domino Web server plugin we just installed. The service upgrade program can be downloaded from the support area of the WebSphere Web site:

ibm.com/websphere

Go to the Support section. Select the product page for WebSphere Advanced Edition and look for support downloads. Be sure to select either advanced or standard edition, depending on the version of WebSphere you have installed. The service upgrade file we used was named *Fixpack_Adv_3021_NT.zip*.

*** Installation Note***

Be very careful to shut down the WebSphere Console, the WebSphere NT Service, the HTTP server (Domino, IIS or other) if you already have started any of these services. For the IIS server, you can stop it either from the Microsoft Management Console or from the NT services panel.

If this is not done, the service upgrade installation will still proceed and report success, but files in use during the installation will not be updated even though the installation will report success. The resulting mixed configuration will give errors when trying to use the Domino or IIS plug-ins.

- Ensure that the system CLASSPATH variable points to the JDK1.1.7p Class.zip file as its *first* entry, as described in 2.5.1, "Editing the system CLASSPATH to include the JDK" on page 20.

The reason this does not need to be done for WebSphere itself is that WebSphere stores environment information in its *admin.config* file and does not reference the system CLASSPATH variable. However, the service upgrade installation uses a system JVM which relies on the system CLASSPATH. If this is not set correctly, the JVM cannot initialize since it needs basic Java classes such as Threading support to start.

- Unzip the service upgrade file to a temporary directory and install it by running the **install.bat** file from a command prompt.

The installation program will prompt you for the absolute path to the WebSphere installation directory. By default, the installation path is:

`X:\WebSphere\AppServer`

where X is the drive on which you installed WebSphere. Further instructions are in the Readme file in the installation directory. Press Enter to start the upgrade.

Note: The Readme implies that the classpath can be set manually by running Java from the command line and specifying it; however, we were unable to get this to work. This is why we emphasize editing the CLASSPATH manually as described in 2.5.1, “Editing the system CLASSPATH to include the JDK” on page 20 and shown in Figure 13 on page 21.

Once the service upgrade has been installed, it is necessary to restart Windows NT to commit the changes.

Finally, if you want to use the IIS Web Server, PTF PQ37562 should be installed by unzipping the file (obtained from the same site as the WebSphere service upgrade) and copying the iis20.dll contained in it to the WebSphere bin directory. If this is not done, WebSphere servlet support will not work properly using IIS as the HTTP server.

After restarting your computer you are now ready to start the WebSphere server.

Note that if you did not stop all services (HTTP servers, WebSphere itself and Domino) you may experience problems when you try to access WebSphere through a browser; we discuss this in 2.7.4.2, “Symptoms of an incomplete service upgrade installation” on page 44.

2.7.3 Starting and running WebSphere

WebSphere Application Server is installed as a Windows NT service, which by default must be loaded manually. This is most convenient for testing but should be changed to *automatic* for production.

Start the WebSphere server from the Services application with the following steps.

1. Open the Windows NT Control Panel by selecting:

Start -> Settings -> Control Panel

2. Double-click **Services** in the Windows NT Control Panel. The Services window is shown in Figure 26.

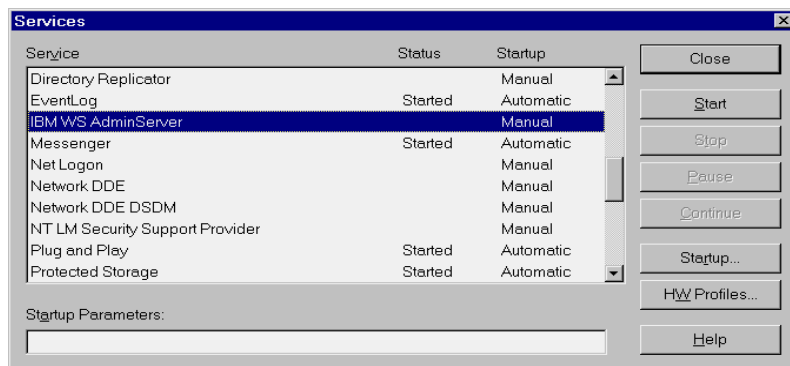


Figure 26. Starting WebSphere from the NT Services panel

3. Scroll down and select **IBM WS AdminServer**. Then click the **Start** button.

Confirmation that the service is starting will be shown in an informational popup as shown in Figure 27.

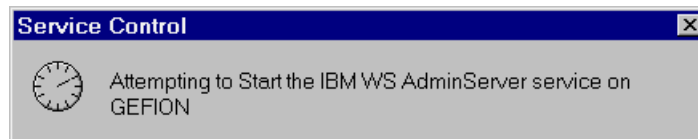


Figure 27. Startup confirmation for WebSphere NT service

This may take several minutes, especially when the server is first set up and the WebSphere WAS database is created. The WebSphere service can also be stopped from the Windows NT Services panel.

Close the Services panel once the WebSphere server has been started.

Note: If you get an error while attempting to start the WebSphere server look at the troubleshooting information on the WebSphere Web site.

Look for a Frequently Asked Questions (FAQ) document for WebSphere Application Server in the Support area. For example, if you get an *NT error 2140* you can read in the FAQ that this means WebSphere couldn't connect to the database (DB2). Some possible reasons for this are that paths to the Domino files weren't specified correctly or the DB2 Java files haven't been added to the CLASSPATH.

2.7.3.1 Using the administrator's console to start the default server

The WebSphere AdminServer service has now been started for the first time. However, it is not yet ready to run any Java components since no JVM capable of running WebSphere applications has been started. This is done by using the WebSphere administrative console.

1. Select **Start -> Programs -> IBM WebSphere -> Application Server V3.0 -> Administrator's Console**.

This will start a command shell followed by a startup screen of a standalone Java application.

Wait for the console to initialize completely. Once the icon on the lower left has finished rotating (or the message "Console Ready" is displayed in the lower message pane), it is ready to accept commands.

2. Select the Topology tab in the left pane and expand the **WebSphere Administrative Domain** by clicking on the + sign next to it. The host name of the server (in our case *gefion*, your host name will be different) should be listed.

Expand the view of the host name node by clicking on the + sign next to it and expand the node **Default Server**. This can be expanded in turn to display the *DefaultContainer* and the *servletEngine*.

3. Right-click the Default Server. The administrative console will now look similar to that shown in Figure 28 on page 38.

Select **Start** on the popup menu (or simply select Default Server by clicking on it and press the large green button in the button bar).

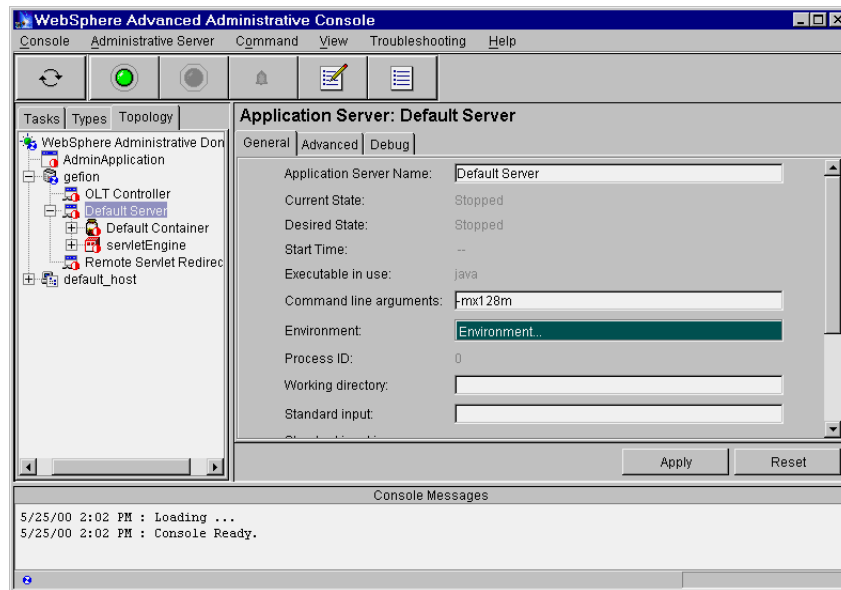


Figure 28. WebSphere Advanced Administrative Console

Confirmation of the load process is shown by a progress bar in the lower right. Once the JVM is loaded and the default server is started, an informational popup is displayed as shown in Figure 29.

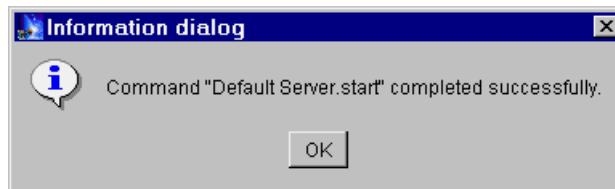


Figure 29. Default Server startup confirmation message

Note: Once the server is started manually, its state is stored in the WebSphere configuration database (the WAS database). The administrative server (the NT service started earlier) will then read its state and re-start the default server if it is stopped outside the administrative console for any reason (such as rebooting the machine or a system failure).

Note that the administrative console will indicate the state of the components started, as shown in Figure 30 on page 39.

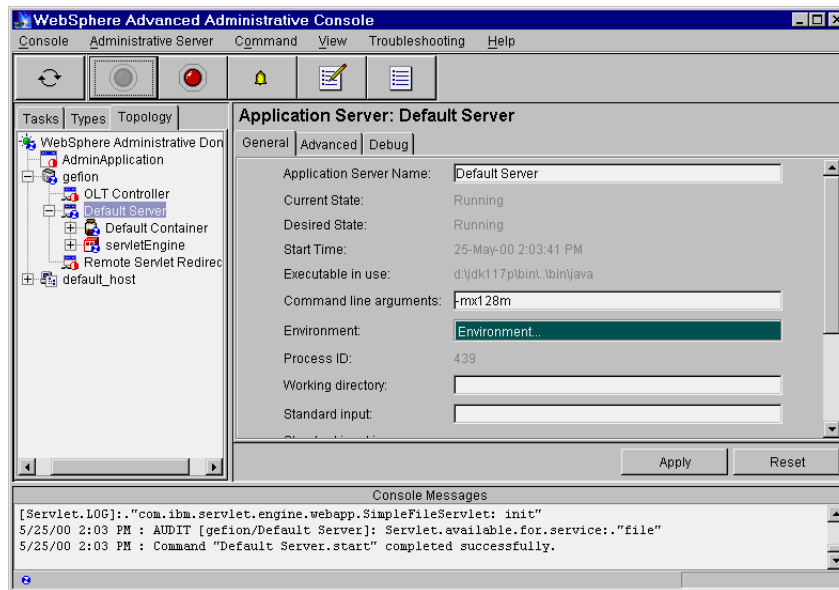


Figure 30. WebSphere Administrative Console showing components started state

The changes are:

- The green start button in the button bar is greyed out and a red stop button is highlighted. You can use this button to stop the default server.
- The symbols beside the components change from a red *stopped* symbol to a blue *running* symbol

The servlet server is now ready and you can close the administrative console.

2.7.4 Installing the WebSphere DSAPI plug-in for Domino R5

The WebSphere installation modifies the httpd.cnf file for the Domino Web server to include some lines that point to a file named go46.dll. This file was used for Domino and WebSphere integration in earlier versions of the products. When the DSAPI plug-in for Domino is installed, the changes in httpd.cnf and the go46.dll file are no longer needed.

To clean up the Domino httpd.cnf we can comment out the lines added by the WebSphere installation, as shown in Figure 31 on page 40, to disable the go46.dll plugin. Do not comment out the two service/cgi-bin lines since they are not related to WebSphere. You can also simply copy the httpd.bak file created by WebSphere installation to httpd.cnf, since this file is the preinstallation backup copy of the http.cnf file.

```

# Server-side imagemap support
#
#####

#NameTrans * D:\WebSphere\AppServer\bin\go46.dll:nametrans_exit
#Authorization * D:\WebSphere\AppServer\bin\go46.dll:authorizat
service /cgi-bin/htimage* INTERNAL:HTImage*
service /cgi-bin/imagemap* INTERNAL:HTImage*
#Service IBMWebSphere D:\WebSphere\AppServer\bin\go46.

#####
# EOF
#ServerInit D:\WebSphere\AppServer\bin\go46.dll:init_exit D:\Web
#ServerTerm D:\WebSphere\AppServer\bin\go46.dll:term_exit
#Pass /IBMWebAS/samples/* D:\WebSphere\AppS
#Pass | /IBMWebAS/* D:\WebSphere\AppServer\wel

```

Figure 31. Disabling the go46.dll plugin

Once the httpd.cnf file has been edited or copied (you may wish to back it up if you edit it), the DSAPI plug-in can be installed in Domino using the following steps:

1. If the Domino Server isn't running, start it with this sequence:
Start -> Programs -> Lotus Applications -> Lotus Domino Server
Your menu may be slightly different if you chose another program folder for Domino.
2. Open the Domino Administration Client.
Make sure you are logged on with a Notes ID that has Administrator rights to edit the Domino Directory.
 - Select the server that Domino and WebSphere are installed on.
 - Select the **Configuration** tab.
 - Open the **Current Server Document**.
 - Select the **Internet Protocols** tabs and then the **HTTP** sub tab.
3. Click the **Edit Server** action button and go to the **DSAPI** section on the right side of the panel.
4. Enter the absolute path of the WebSphere Domino R5 plug-in, as shown in Figure 32 on page 41. In our case the path was:
D:\WebSphere\AppServer\bin\domino5.dll
5. Click the **Save and Close** action button. The DSAPI plug-in will be loaded when the Domino HTTP task is next loaded.

Note: The server document allows specification of multiple DSAPI plug-ins; the order in which they are entered will define the order in which they are called. We did not test multiple DSAPI plug-ins in our testing, however.

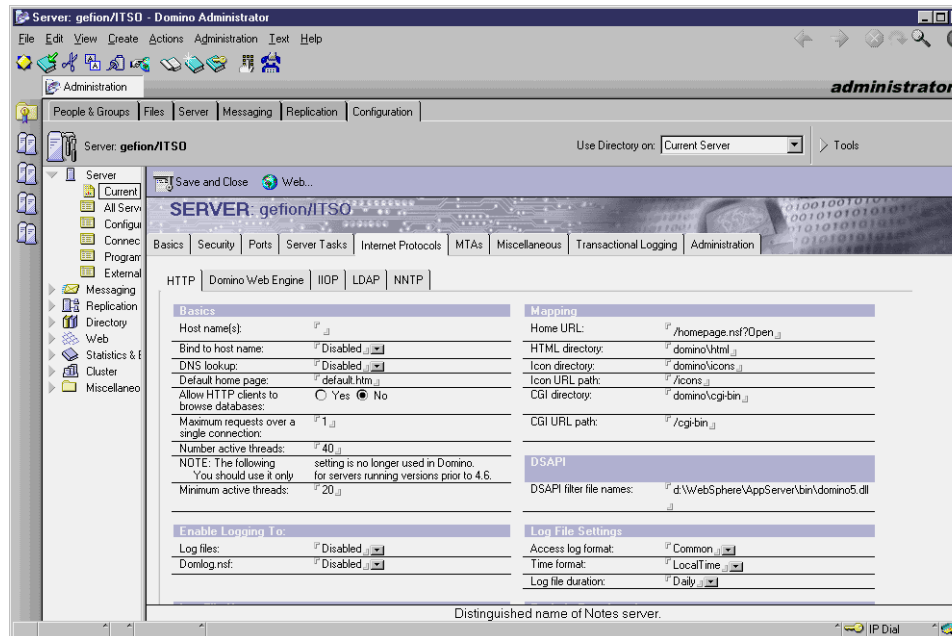


Figure 32. Installing the WebSphere Domino 5 DSAPI plugin

Once these configuration changes have been made, recycle the HTTP task by issuing the following command at the Domino server console:

```
tell http restart
```

Note that it is not necessary to stop any WebSphere component or the Domino server, only the Domino HTTP task. Once the Domino R5 plug-in has been installed, it will put a message on the Domino server console (and in the log file) when it is loaded and unloaded. This is illustrated in Figure 33 on page 42.

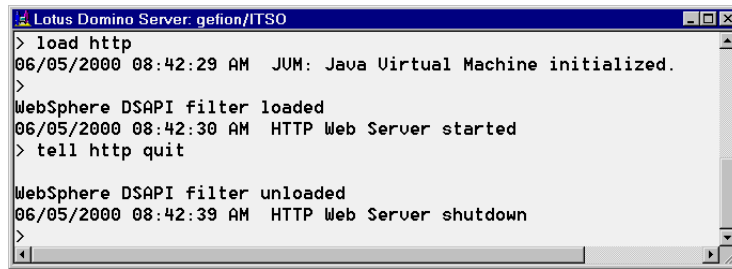


Figure 33. Console Messages confirming installation of Domino 5 WebSphere plugin

2.7.4.1 Verifying servlet support

To test that the servlet support is working, start your Web browser on the server machine and point it to the following URL:

`http://localhost/servlet/snoop`

Note: Java distinguishes between upper and lower case. Make sure you write the servlet name exactly as it is (in this case all lower case). Using an upper case letter specifies a servlet that does not exist and you will get an Error 500 together with a very long message (starting with “ClassNotFound”) returned to your browser because the specified servlet isn’t found.

The result of a successful invocation of the snoop servlet is shown in Figure 34.

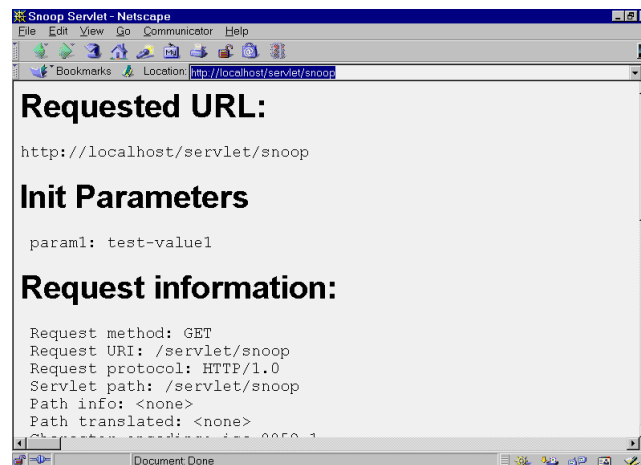


Figure 34. Running the sample servlet snoop to confirm WebSphere-Domino installation

If you try it from another computer, replace *localhost* in the URL with the IP name of the computer like this:

<http://gefion.lotus.com/servlet/snoop>

This sample servlet simply returns the original request to the browser; this confirms that the request has been routed through Domino to WebSphere and returned.

At this point Domino and WebSphere are working together, although Domino is only acting as an HTTP server for WebSphere at this stage.

Note that there are other parameters in the server document that can be set in the tab Domino Web Engine under Internet Protocols. The Domino Administration Help suggests setting the field *Java Servlet Support* to *Third Party Support* to enable WebSphere servlets. Our experience is that servlets can be loaded no matter what this field is set to, including the default *None*. However, specifying Third Party Support does allow Domino to load a JVM for Java agents while still using WebSphere for servlet support. The reason that the setting does not affect servlet processing is that the DSAPI filter is called immediately after receiving an HTTP request before any Domino processing takes place. Although the original intention of the DSAPI exit was to enable third party or custom authentication, it also serves as a pre-emptive exit for plug-ins such as the WebSphere Domino5.dll. We illustrate setting **Java Server support** in Figure 35.

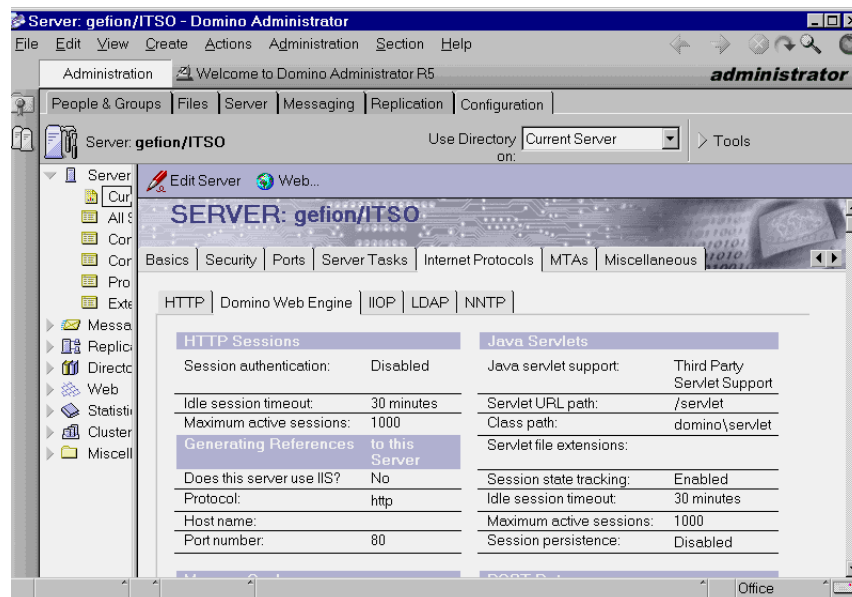


Figure 35. Setting the Java Servlet Support field in the Domino Server document

The Domino server can be started before or after the WebSphere Application Server. Our recommendation is to start it after WebSphere so that requests for servlets will not be forwarded to WebSphere before it starts. However, if WebSphere is using Domino as its LDAP authentication directory, Domino *must* be started first since the WebSphere administrator's console binds to the LDAP directory using the administration user ID as part of its startup process.

2.7.4.2 Symptoms of an incomplete service upgrade installation

Our testing showed that, if the WebSphere service upgrade was incompletely applied, we were unable to use Domino R5 as an HTTP server to reach WebSphere. If we tried using the DSAPI plug-in (domino5.dll), the WebSphere server logged error messages in its trace log:

2000 - Error Invalid Service Request. No Read Callback provided.

The message returned from a (NetScape) browser is shown in Figure 36:

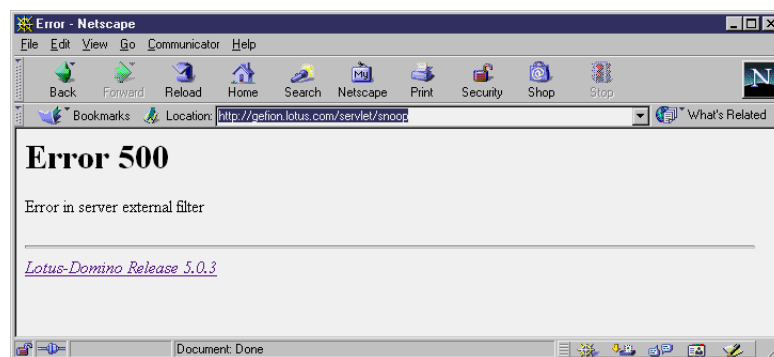


Figure 36. Error Returned from Netscape Browser using Domino 5.0.3 and WebSphere 3.02.1

Finally, the message shown on the Domino console if the DSAPI plug-in is used is shown in Figure 37.

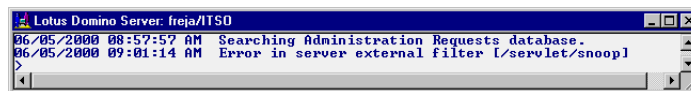


Figure 37. Domino Console message using Domino R5.0.4 and WebSphere 3.02.1 and DSAPI plug-in

If you see these error messages, uninstall the service upgrade and reinstall it, ensuring that all WebSphere, Domino and IIS processes have been shut down.

2.8 Using the IIS HTTP Web Server in place of Domino

Domino and WebSphere applications can both be utilized even if a Web server other than the Domino HTTP server is used. WebSphere supports several different HTTP servers (the current list is shown in Figure 23 on page 32). One of them is Microsoft Internet Information Server (IIS), which Domino R5 also can use as its HTTP server. Today, Domino supports IIS and its own HTTP server; in a future release it will support the Apache HTTP server as well. This also implies that it will support the IBM HTTP server which is based on the Apache server. Today, however, the only substitute HTTP server which supports both Domino and WebSphere is IIS.

In Appendix A, “Using the IIS Web Server for Domino and WebSphere” on page 205 we walk through the steps to install Microsoft IIS 4.0 to act as HTTP server for Domino and WebSphere.

2.9 Different Web server plug-in behavior

During our testing we enabled trace level logging of the Web server plug-ins we used and discovered that the plug-in for the Domino HTTP Web server forwards *all* HTTP requests to WebSphere. If the request is for Domino the request is then returned from WebSphere to Domino. On the other hand, if we use the IIS plug-in then Domino requests are forwarded directly from IIS to Domino while servlet requests are forwarded directly to WebSphere. For more details about this see Appendix B, “WebSphere plug-in behavior and tracing options” on page 217.

2.10 Summary

In this chapter we have covered the steps necessary to install Domino R5 and WebSphere 3.02 on a Window NT machine.

Chapter 3. The elements of WebSphere applications

The WebSphere application server is available in three different editions. Each of them enables you to enhance your Web applications with certain elements.

- The *standard edition* provides you with an open, standards-based, Web server deployment platform. It enables you to use Java servlets and Java Server Pages (JSPs) and provides connections to back-end database systems.

WebSphere standard edition is included with Domino R5 Advanced and Enterprise server.

- The *advanced edition* adds the support of Enterprise Java Beans (EJBs) and provides better support for multi-server environments.
- Along with the above features, the *enterprise edition* includes the IBM Component Broker, which adds support for CORBA and the IBM TxSeries that provides a transactional application environment.

The WebSphere application server is part of the foundation of IBM's platform for e-business. On top of this, IBM offers a range of foundation service products in areas like performance optimization, site analysis, security, and so on. In Chapter 7, "Scalability and redundancy with Domino and Websphere" on page 179 we discuss a set of products called the WebSphere Performance Pack and how Domino works together with those products, but we primarily concentrate on the WebSphere application server and Domino.

In this chapter we introduce you to the following common components in a WebSphere transactional application:

- Servlets
- JavaServer Pages
- Enterprise JavaBeans

In Figure 38 on page 48 you can see how the server-side components of transactional WebSphere applications play together.

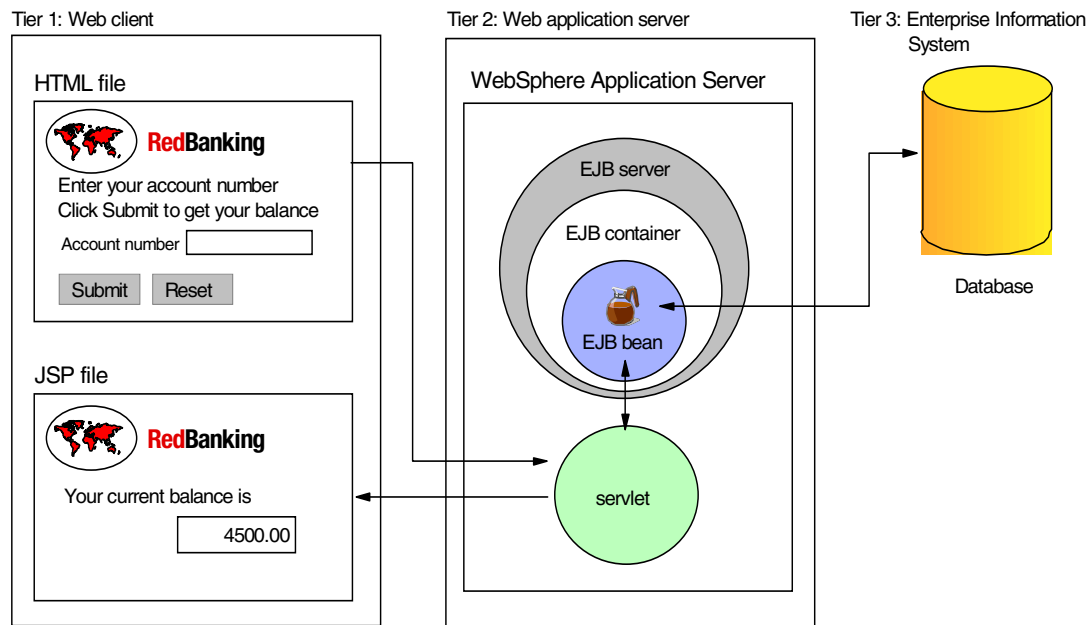


Figure 38. Use of servlets, EJBs and JSPs in a WebSphere transactional application

Since server-side applications are typically 3-tiered, the components are spread across the tiers as shown in the figure. The components from left to right are as follows:

- Tier 1: This tier is the user's view. From a system perspective, it is the client view. The user contacts a server-side application over the Web through a Web browser (containing an HTML file served by a HTTP server, in our case Domino). The HTML file is linked to a servlet in tier 2 at the server. The user sees information returned from the server in an HTML file generated from a JSP file.
- Tier 2: This tier is where the business logic resides at the server. The application's processing takes place here, typically at a high-end server. The EJB bean containing the non-visual functions resides in a container that in turn resides in a Web application server. In our case, the server is WebSphere Application Server. The container ensures persistence for those EJBs that are of type entity beans; that is, like a unit of work in a database transaction, containers ensure that a function is either completed or rolled back in order to maintain application integrity.
- Tier 3: This tier provides the services that the server-side application accesses. In the figure the services are in a database system, but they

could include many other resources, such as CICS records, MQSeries messages, SAP data, and IMS databases. These services are collectively referred to as the Enterprise Information System. Since there can be many services, a complex server-side application can have many tiers.

Another way of imagining the three tiers is to see them as layers. The first layer contains the user interface. The second layer contains the application programs. The third layer contains the services available to the application programs. The layers are loosely coupled, meaning they have little dependency on each other.

Note: When you add Domino functionality to your solution you may chose to use Domino forms for both input and presentation of results in the client side.

We now go a bit more into detail with Java servlets, Java ServerPages and Enterprise JavaBeans.

3.1 Java servlets

Servlets are small Java programs that run on the Web application server. They are portable across platforms and across different Web servers. They usually interact with the servlet engine running on the Web application server through HTTP requests and responses, which are encapsulated as objects in the servlet.

A servlet is loaded by the Web application server, and remains loaded across client requests. This means that the servlet can maintain system resources, like a database connection, between requests.

Servlets extend the `javax.servlet.http.HttpServlet` class. Usually the actions the servlet performs are implemented in one or more of the following methods:

- `init ()` - This method is called when the servlet is loaded. It is not called again if the servlet URL is called for the second or subsequent time.
- `doGet (HttpServletRequest req, HttpServletResponse resp)` is called if the servlet is invoked via an HTTP GET request. This method is called when a servlet is called by entering its URL in a browser.
- `doPost (HttpServletRequest req, HttpServletResponse resp)` is called if the servlet is invoked via an HTTP POST request. This request usually is sent using a form. For information on how to call a servlet from a Domino R5 form, see 5.1.3, "Posting data to servlets from Domino R5 forms" on page 111.

The `HttpServletRequest` parameter contains attributes of the request for the servlet. You can use the `HttpServletResponse` to generate a response the servlet engine is sending to the requester after the servlet has performed its tasks.

The documentation for the whole java servlet API is available on:

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

More information about the deployment of servlets in WebSphere is available in 5.1.1, “Servlet URLs” on page 105.

To a Domino Web programmer, servlets in many cases are preferable to Web agents. This is because servlets, once loaded, stay in memory, while Domino Web agents have to be loaded and unloaded for every invocation. This makes applications that use servlets instead of agents more scalable. From a Domino perspective you can call servlets “agents on steroids.” However, servlets are not as well integrated with Domino as agents are. In a servlet you do not have the context of who the Domino user is, and the granular Domino security does not apply to servlets. When common authentication among Domino and WebSphere is being implemented in the products, you will be able to get some Domino user context in a servlet that is being served by Domino, but not as rich as the context you get in a Domino agent.

Domino R5 also has its own servlet engine, so you do not even need WebSphere to include servlets in a Domino-based application, but we only discuss the WebSphere servlet engine in this book.

3.2 JavaServer Pages

JavaServer Pages (JSPs) are an easy way to combine Java code with HTML, which means data access programs with layout. In Web applications the response sent to the client is often a combination of static page design and dynamically-generated data. In this situation, it is much easier to work with JSPs than with servlets that use the `HttpServletResponse` class to construct Web pages.

JSPs are HTML files containing additional tags. WebSphere supports versions 0.91 (default value for its `default_app` during installation) and 1.0 of the JSP specification and adds some specific tags. You can place Java code between some of the JSP tags. The most important tags that allow Java code are:

- Import statements for Java packages. The packages are available on the whole page. Example:

```
<%@ language="java" import="java.sql.*" %>
```

- **Declaration:** declares a variable or method. Example:

```
<%! int iStatus = 0; %>
```

- **Expression:** contains one expression. Example:

```
<%= request.getParameter("message") %>
```

- **Scriptlet:** contains a code fragment, that is, a short Java program.

Example:

```
<% for (int iCounter = 0; iCounter < vejbAccounts.size(); iCounter ++) {  
    AccountDataBean ejbAccount =  
        (AccountDataBean) vejbAccounts.elementAt(iCounter);  
    int iAccountID = ejbAccount.getID();  
}%>
```

In addition, you can include HTML pages, servlets, and EJBs into a JSP. You can find a description of all tags at:

<http://java.sun.com/products/jsp/tags/11/tags11.html>

The first time a new or modified JSP is called, WebSphere generates a temporary servlet. This servlet is called every time the browser calls the JSP. You can see how this works in Figure 39.

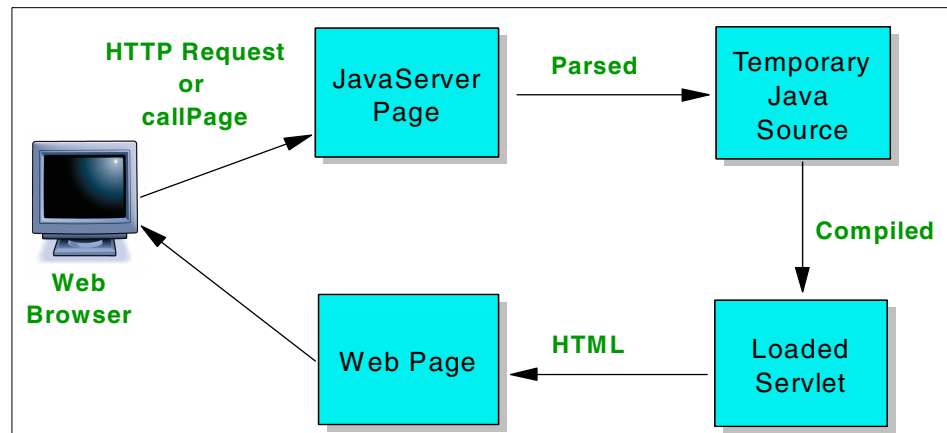


Figure 39. How Java ServerPages work

The deployment of JSPs in WebSphere is described in 5.2.1, “JavaServer Page URLs” on page 115.

In some ways, JSPs are similar to the Active Server Pages (ASP) technology from Microsoft. The difference is that the scripting language is Java in JSPs

while it is Visual Basic in ASPs. In the future WebSphere also will support LotusScript in JSPs, making it more attractive for existing Domino programmers with LotusScript skills to use this technology. It is also worth noting that JSPs in many cases are used to display dynamic result pages, as shown in Figure 38 on page 48. If you have an application that includes Domino and WebSphere elements, there may be situations where it actually makes more sense to use Domino forms to display the dynamic results being returned from servlets.

3.3 Enterprise JavaBeans

Enterprise JavaBeans is Sun's trademarked term for their EJB architecture (or "component model"). When writing to the EJB specification you are developing *enterprise beans* (or, if you prefer, *EJBs*).

Enterprise JavaBeans are designed to be installed on a server, and accessed remotely by a client. The EJB framework provides a standard for server-side components with transactional characteristics.

An EJB client program can be any program that can communicate via the Java protocol Remote Method Invocation (RMI) or via the Internet Inter-ORB Protocol (IIOP). RMI is only possible if the client program is written in Java.

A description of how to write EJBs for WebSphere is available at:

http://www-4.ibm.com/software/webservers/appserv/doc/v30/ae/web/doc/ent_beans/atswpg00.pdf

The EJB framework specifies clearly the responsibilities of the EJB developer and the EJB container provider. The intent is that the "plumbing" required to implement transactions or database access can be implemented by the EJB container. The EJB developer specifies the required transactional and security characteristics of an EJB in a deployment descriptor (this is sometimes referred to as declarative programming).

For an example of how to deploy EJBs in an EJB container in WebSphere refer to 4.3, "Deploying the IBM WebSphere account example" on page 67.

There are two types of Enterprise JavaBeans:

- Session
- Entity

3.3.1 Session beans

A typical *session bean* has the following characteristics. It:

- Executes on behalf of a single client.
- Can be transactional.
- Can update data in an underlying database.
- Is relatively short-lived.
- Is destroyed when the EJB server is stopped. The client has to establish a new session bean to continue computation.
- Does not represent persistent data that should be stored in a database.
- Provides a scalable runtime environment to execute a large number of session beans concurrently.

For example, the task associated with transferring funds between two bank accounts can be encapsulated in a session bean.

3.3.2 Entity beans

A typical *entity bean* has the following characteristics. It:

- Represents data in a database.
- Can be transactional.
- Shares access from multiple users.
- Can be long-lived (lives as long as the data in the database).
- Survives restarts of the EJB server. A restart is transparent to the client.
- Provides a scalable runtime environment for a large number of concurrently active entity objects.

Typically, an entity bean is used for information that has to survive system restarts, in contrast to session beans where the data is transient and does not survive when the client's browser is closed. For example, the information about a bank account can be encapsulated in an entity bean.

3.3.2.1 Bean or Container Managed Persistence

An important design choice when implementing entity beans is whether to use *Bean Managed Persistence* (BMP), in which case you must code the JDBC logic, or *Container Managed Persistence* (CMP), where the database access logic is handled by the EJB container.

The business logic of a Web application often accesses data in a database. EJB entity beans are a convenient way to wrap the relational database layer in an object layer, hiding the complexity of database access. Because a single business task may involve accessing several tables in a database, modeling rows in those tables with entity beans makes it easier for your application logic to manipulate the data.

Another big advantage of entity beans with Container Managed Persistence is that they provide a standard-based access to database systems. You can change the EJB container, the database system or the database without changing the code of the EJB. Only the deployment descriptor of the bean must be changed.

If you are using a database system that is not supported by your Web application server, you write entity beans with Bean Managed Persistence. For example, WebSphere does not support CMP using Domino databases as the persistent store. If you want to store entity beans in Domino R5, you have to use BMP. For entity beans that must implement their persistence themselves, the code is more complicated than that for entity beans with CMP. In addition, the code of entity beans with Bean Managed Persistence often uses special features of the database system. The database system cannot be changed without changing the EJB code.

3.3.3 EJB architecture

We will go one level deeper in our discussion of EJBs to give you a better understanding of our examples involving EJBs. Figure 40 on page 55 shows a diagram of client-to-EJB interaction in more detail.

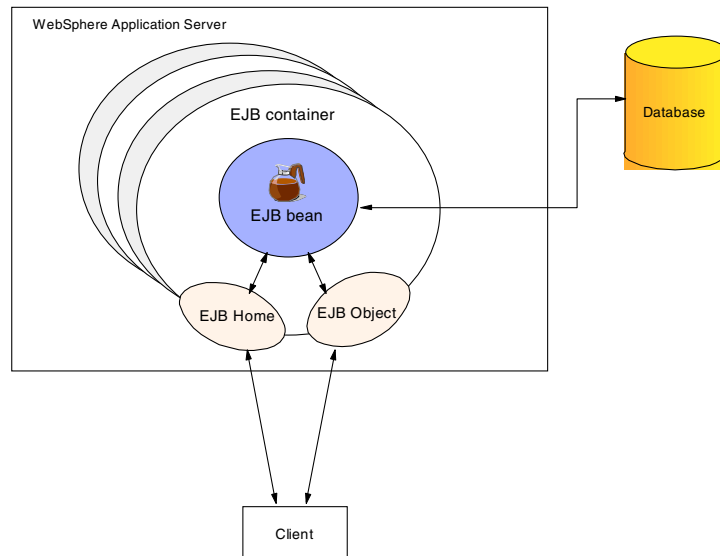


Figure 40. EJB interaction detail

In order for the management of EJBs to be handled by the server properly, a client must access EJBs only through a proxy provided by the EJB container. This allows the container to control persistence, security, caching and connection management with no knowledge by the client that all of these functions are occurring with no code in the EJB to control these functions. In order to facilitate this, all client access to EJBs is done by means of instances of the *EJB Home* and the *EJB Object* interfaces, which are created by the developer during development. (The *EJB Object* interface is sometimes also called the remote interface.) This allows the server to perform management tasks under the covers by mapping the calls to these interfaces to appropriate calls to the EJB itself, and also by calling infrastructure methods on the EJB to control transactions and storage to databases.

The *EJB Home* interface instance is responsible for allowing clients to find and create EJBs. For entity beans the home interface includes methods for finding single beans or groups of beans based on certain criteria, including at least one method that allows the location of a bean in a database using a primary key class. For both entity and session beans the home interface includes methods to create new instances of an EJB inside the container and return a reference to an *EJB Object* interface instance for the bean. Note that there is one *EJB Home* interface instance per class of EJB in a container, but there may be many *EJB Object* interface instances depending upon how many actual instances of the EJB class are present.

The EJB Object interface is responsible for providing access to the operations of an EJB. Each call to an EJB Object interface instance is mapped to a corresponding call to a bean instance by the container, subject to security considerations. Because of the separation from the actual bean, the container is free to release resources used by the bean (such as database connections or even the bean instance itself) to other uses, and restore the EJB instance when a call is made to it by a client.

3.3.3.1 Steps in using an EJB

Figure 41 shows the steps involved in a client accessing an EJB.

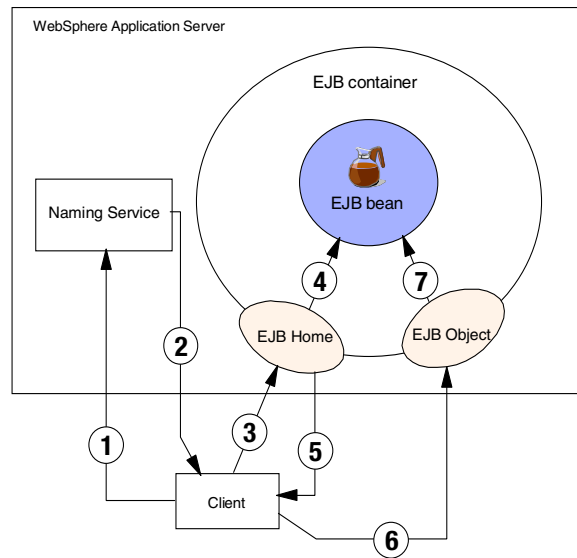


Figure 41. Steps used to connect to an EJB from a client program

The sequence of actions that occur when a client program wants to use an EJB are as follows:

1. The client requests from the naming service (provided as one of the components of WebSphere) a reference to the EJB Home interface of a particular class of EJBs.
2. The naming service replies with the location of the Home interface instance for the EJB class in the container in which the EJB is deployed.
3. The client performs either a create (for a new bean instance) or a find (for an existing entity bean instance) on the EJB Home interface instance.
4. The EJB Home interface instance locates or creates the EJB instance and places it in the container, and creates the EJB Object interface instance.

5. The EJB Home interface instance replies to the client with a reference to the EJB Object instance.
6. The client calls methods on the EJB Object interface instance to access business logic on the EJB.
7. The EJB Object interface instance calls the corresponding methods on the EJB while the container manages the resources needed to accomplish this task.

The latest EJB specification is 1.1. The most significant changes from EJB 1.0 are the use of XML-based deployment descriptors and the need for vendors to implement entity bean support to claim EJB compliance. WebSphere Advanced and Enterprise Server 3.02 that we worked with in preparing this book support the EJB 1.0 specification.

3.4 Summary

In this chapter we have given a brief overview of the components in a WebSphere transactional application to prepare you for the following chapters, where we describe how you as a developer can make Domino and WebSphere interact. We have covered:

- Servlets
- Java ServerPages
- Enterprise JavaBeans

Chapter 4. Accessing Domino R5 from WebSphere

In this chapter we first explain how to install and set up IBM VisualAge for Java, to enable you to follow our examples in this chapter and in Chapter 5, “Using WebSphere from Domino R5” on page 105.

Then we discuss the following methods of accessing Domino R5 from WebSphere:

- Using the Lotus Domino Driver for JDBC:
 - From a servlet
 - From a JavaServer Page
- Using the Domino Object Model to send mail from:
 - A servlet on the same machine
 - A servlet via IIOP
 - A JSP via IIOP
 - An EJB via IIOP

4.1 Setting up IBM VisualAge for Java for the examples in this book

For most of the techniques to access Domino from WebSphere or to access WebSphere from Domino, you have to create Java code. We assume that you know how to develop programs using Java.

In all our examples we use IBM VisualAge for Java (VA Java). Of course, you can create all the Java code using the development environment of your choice, but if you are not an experienced Java developer it is easier for you if you install VA Java. This enables you to follow our instructions for creating the examples exactly.

All the examples we created in VA Java in this chapter are available at the IBM Redbooks Web site. See Appendix C, “Additional Web material” on page 231 for more information.

4.1.1 Installing VA Java

First you install the professional edition or the enterprise edition of VA Java. The professional edition comes with WebSphere. You cannot use the free entry edition, since this version is limited to 750 classes and the WebSphere and Domino R5 support consists of more than 750 classes. We used VisualAge for Java, Professional Edition for Windows, Version 3.0.

We will not give a step-by-step description of how to install VA Java. If you are in doubt during the installation, simply use the default setup values.

After completing the installation, start VA Java, select **Go to the Workbench** and click **OK**.

4.1.2 Adding features to VA Java

The classes you need to create servlets and access WebSphere are not imported into the workbench during the installation of VA Java. You must add these features using the following steps:

1. Select **File - Quick Start**.
2. Select **Features** and **Add Feature** as shown in Figure 42; click **OK**.

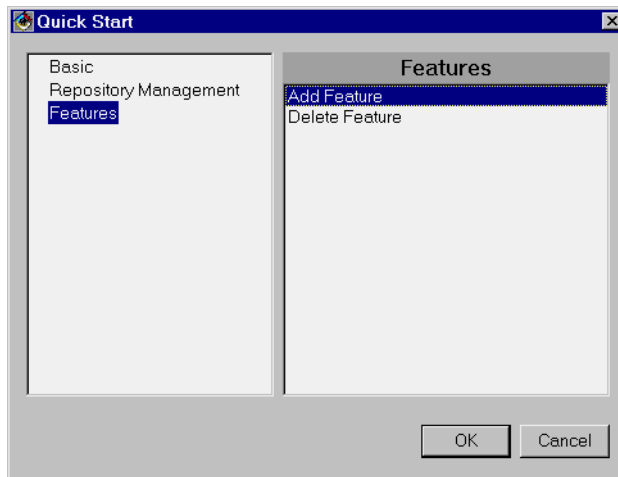


Figure 42. Adding features to VA Java

3. In the list box select **Sun Servlet API 2.1** as shown in Figure 43 on page 61; click **OK**.

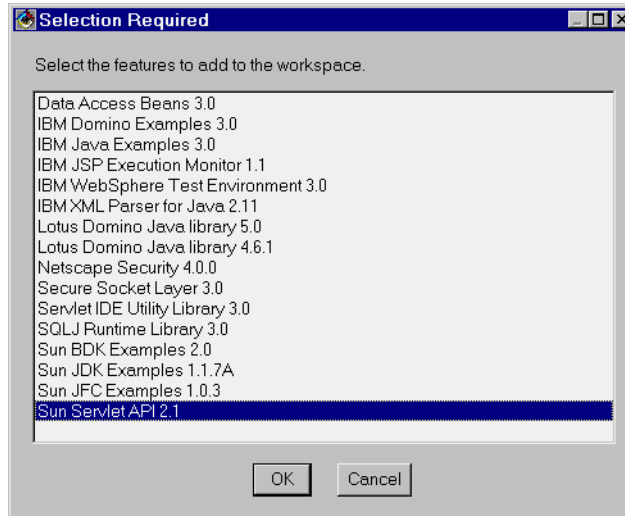


Figure 43. Selecting a feature to be added to VA Java

Now you have added the code necessary to develop servlets.

4. Add the **IBM WebSphere Test Environment 3.0** just like you added the Sun Servlet API.

The IBM WebSphere Test Environment enables you to test and debug your servlets and JSPs before you put them into production. If you use the enterprise edition of VA Java the test environment also supports testing and debugging of Enterprise Java Beans.

Now we add support for Domino R5 to VA Java.

4.1.3 Importing the Domino R5 classes

We are now going to import the Domino R5 classes in VA Java, but first we need to decide which file to import from. It depends on whether we will access Domino on the same machine, from a machine that has the Notes client installed, or from a machine with neither. You have the following package options for the Domino Java Library:

- *Notes.jar*

Notes.jar contains the classes needed for backward compatibility with Domino R4.6. It also contains the high-level *lotus.domino* package and the *lotus.domino.local* package for local calls. Use this package if Domino or Notes resides on the same machine as your Java program

- *NCSOW.jar*

NCSOW.jar contains CORBA support for Domino when used with WebSphere V3. It contains the high-level *lotus.domino* package and the *lotus.domino.corba* package for remote calls.

There is also a *NCSO.jar* file that contains the same packages for remote access, but it cannot be used from WebSphere because it isn't compatible with the WebSphere V3 CORBA support.

In our first example we will access Domino locally and we will thus use *Notes.jar*, as explained a bit further on.

You cannot add the feature called Lotus Domino Java library 5.0 together with the IBM WebSphere Test Environment because VA Java finds that some of the classes this feature contains already exist, and the adding of the feature fails. We need a small work around.

To add the Domino classes do the following:

1. Create a project in VA Java named Domino R5 classes. On the menu select **Selected -> Add -> Project**.
2. Enter the project name `Domino R5 classes` as shown in Figure 44 on page 63.

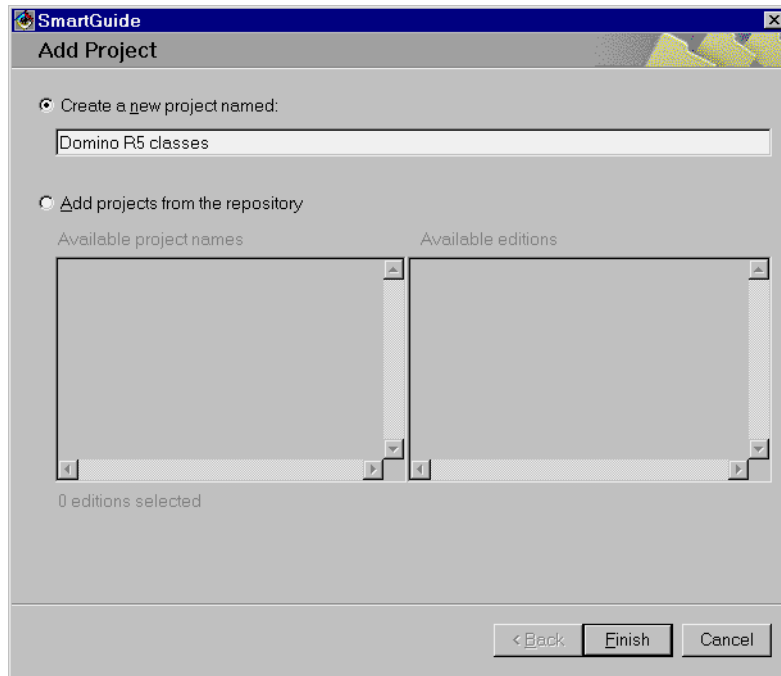


Figure 44. Creating a project in VA Java

Then click **Finish**.

3. After VA Java has created the project you must import the Domino file Notes.jar into this project.
 - a. Highlight the project and select **File -> Import**.
 - b. In the dialog box that appears select **Jar file** and click the **Next >** button.
 - c. Then click the **Browse...** button beside the Filename field and select the Notes.jar file as displayed in Figure 45 on page 64.

The Notes.jar file is in the program directory of your Domino R5 server. In our case it was d:\lotus\domino.

Make sure that **.class** is checked in the dialog box.

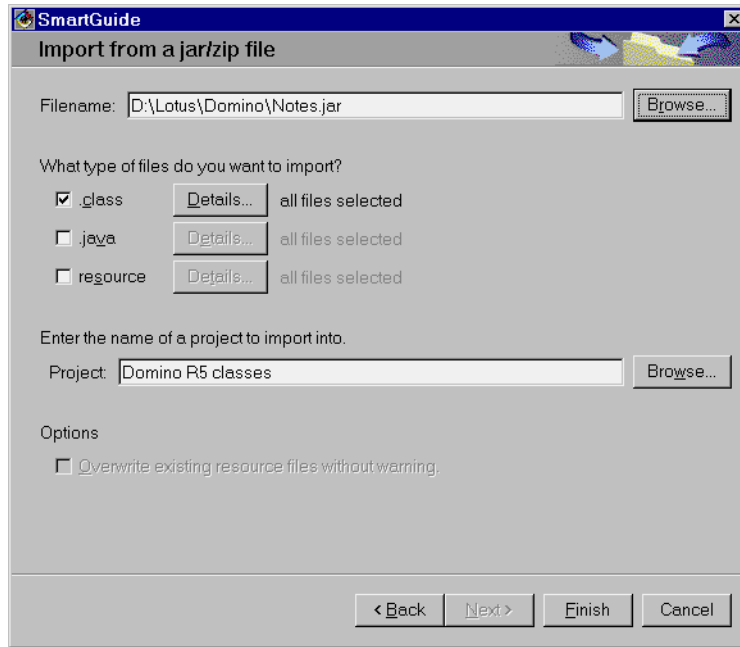


Figure 45. Importing a file into VA Java

d. Click the **Finish** button.

The following informational message box appears:

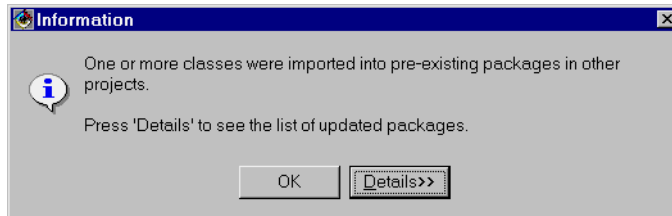


Figure 46. Warning in VA Java after import of Notes.jar

This means that some files already existed in another project. Since Java programs use the package names for importing files, you can ignore this message and click **OK**.

You have now added the Domino support to VA Java.

4.1.4 Importing the IBM account example

For most of our examples we will modify the IBM account example that comes with WebSphere. To import this example do the following:

- Create a project named **IBM Account example**.
- Select **File -> Import**.
 - In the dialog box select **Directory** and click **Next**.
 - Click **Browse** beside the directory field and select the subdirectory
`AppServer\EJBSamples\src\AccountAndTransfer`
of your WebSphere directory.
 - Click **Finish**.

VA Java will import the example code into your new project and you are set up to use the example.

4.1.5 Creating a new package for the example code

We want to separate the code we are writing in our Domino R5 and WebSphere examples from the original example. Therefore, we must create a new package for our code with the following steps:

- In VA Java highlight the **IBM Account example** project and select **Selected -> Add -> Package**.
- Enter the package name `com.ibm.ejs.doc.sg245955`
The package name `sg245955` is taken from the IBM form number for this Redbook to ensure uniqueness.
- Click **Finish**.

We will use this package later on in the book.

4.2 Setting up WebSphere to use Domino

Before you can access Domino from WebSphere you need to specify the paths that enable WebSphere to use the Domino R5 classes and libraries.

We assume that you know enough about WebSphere configuration and the WebSphere administrative console to add or modify the values based on our descriptions below.

4.2.1 Modifying the path WebSphere uses

WebSphere does not use the system entries for the classpath and the library path. You have to set these paths within WebSphere.

4.2.1.1 Library path

To modify the library path you must edit the file `admin.config`. It is located in the `AppServer\bin` subdirectory of your WebSphere directory.

Stop the WebSphere service before editing the `admin.config` file.

Add the Domino R5 program directory to the variable called:

```
com.ibm.ejs.sm.util.process.Nanny.path.
```

Note: You have to double the backslashes in the path specification. In our case we added the following at the end of the path:

```
d:\\Lotus\\Domino
```

4.2.1.2 System-wide class path

The class path can be modified for the whole WebSphere system or for specific WebSphere applications. If you want to use the Domino R5 classes system wide, you must add the file `Notes.jar` that is available in the Domino program directory to the following variable in the `admin.config` file:

```
com.ibm.ejs.sm.adminserver.classpath.
```

In our case we added the following at the end of the path:

```
d:/Lotus/Domino/Notes.jar;
```

After you have saved the `admin.config` file you can start the WebSphere service.

4.2.1.3 Application-specific class path

If you did not include the `Notes.jar` file in the system-wide class path, you must include it into the classpath of the Default Application. That is the Web application we are using in our examples.

- Start the WebSphere administrative console and select the **Topology** tab.
- Expand your node (that is the host name - ours was *gefion*).
Highlight the **Default Server** and stop it using the red button.
- Expand the default server and expand the **servletEngine** below it as well.
- Highlight the **default_app** and select the **Advanced** tab in the pane on the right.

Add the Notes.jar file to the classpath list and click **Apply**.

- Then select the **Default Server** and start it again using the green button.

4.3 Deploying the IBM WebSphere account example

In the examples you will create in this chapter and in Chapter 5 we use and modify an example application that come with WebSphere. It is a simple banking scenario that consists of an entity EJB named *Account*, a session EJB named *Transfer*, and some servlets and other EJBs.

- The Account enterprise bean is a model of a savings or checking bank account. Each account has an unique ID, an account type, and a balance.
- The Transfer enterprise bean models a transfer of funds. It moves a specified amount from one instance of the Account bean to another.

Here we explain how to deploy this example before you can start to modify it.

4.3.1 Create a database for account objects

Since the account EJB is an entity bean, it needs a database. We tested with DB2, but it should be possible to use any database WebSphere supports for Container Managed Persistence (CMP). WebSphere 3.02 supports DB2 and Oracle for CMP. WebSphere 3.5 includes Sybase support for CMP as well. See 3.3.2.1, “Bean or Container Managed Persistence” on page 53 for more information on bean persistence.

Open the administration program of your database system and create an empty database. You do not need to create any tables because WebSphere will do this for you. You can give the database any name you want. We called our database ACCOUNTS.

4.3.2 Install the database driver in WebSphere

You need to install the JDBC driver for your database system. If this is not your first WebSphere application that uses this database system, the driver is already installed and you can omit this step.

To install the driver:

- Click the **Types** tab in the WebSphere administrative console. Right-click **JDBC Drivers**.
- In the menu that appears now, select **Create**.

Enter the name, the implementation class and the URL prefix for the JDBC driver. We used the following values:

- Name
Db2
- Implementation class
com.ibm.db2.jdbc.app.DB2Driver
- URL prefix
jdbc:db2

This is shown in Figure 47.

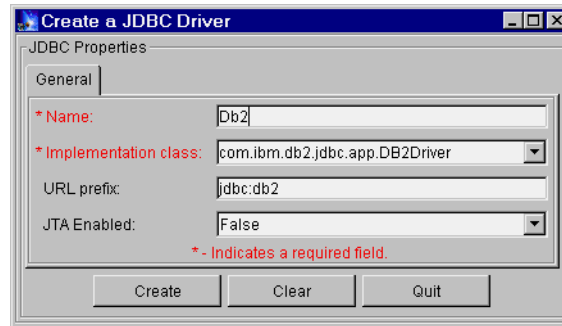


Figure 47. Creating a data source in WebSphere

- Click **Create**
- Click the **Topology** tab.

Right-click the name of the driver you created and select **Install**.

- Select the node you want to install the driver to.

Click **Browse** to select the Java archive that contains the implementation classes of the JDBC driver, as shown in Figure 48 on page 69 where we selected:

D:\SQLLIB\java\db2java.zip

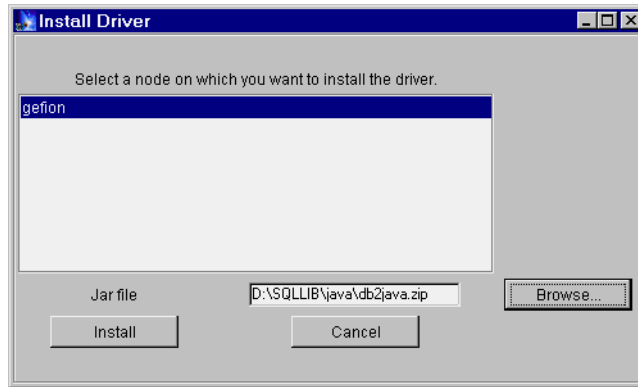


Figure 48. Installing a database driver in WebSphere

- Click **Install** to install the driver.

4.3.3 Create a data source in WebSphere

We will now define the DB2 database we created earlier as a data source in WebSphere.

- Select the **Types** tab in the WebSphere administrative console and right-click **DataSources**.
- Select **Create** and enter the data source name.

The data source name means the logical name in WebSphere. We used the name AccountSource.

Specify the name of the database in the database system. Use the name of the empty database you created in 4.3.1, “Create a database for account objects” on page 67. Our database was named ACCOUNTS.

Select the database driver you want to use. We chose Db2.

Your data source dialog box should look similar to Figure 49 on page 70.

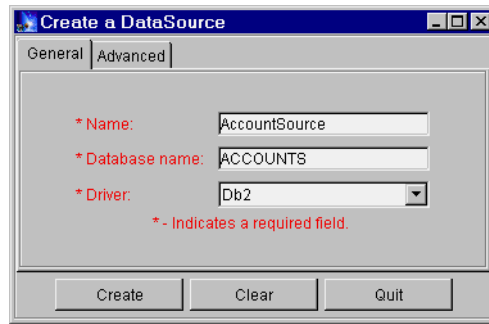


Figure 49. Creating a data source in WebSphere

- Click **Create** to create the data source.

We will now create an EJB container that utilizes the data source we just created.

4.3.4 Creating an EJB container

Between the Account EJB and the database to store the instance data we need an EJB container. The EJB container provides various services to the EJBs it contains. One of them is handling the database interaction.

Create an EJB container like this:

- In the WebSphere administrative console select the **Topology** tab and then open your node and your server.
You can use the **Default Server** or create an new server. We used Default Server.
- Right-click the server and select **Create -> EJB Container**.
- Enter the name you want to use for your container in WebSphere. We called our container Account Sample.

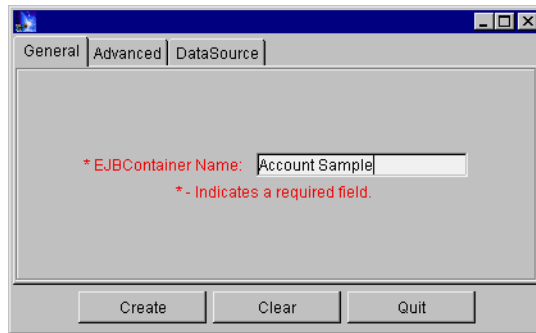


Figure 50. Creating an EJB container in WebSphere

- Click the **DataSource** tab and then the **Change** button.
Select the data source you created in 4.3.3, “Create a data source in WebSphere” on page 69. We selected AccountSource.
- Enter the user name you want to use for the database access (we used db2admin) and the password of this user.

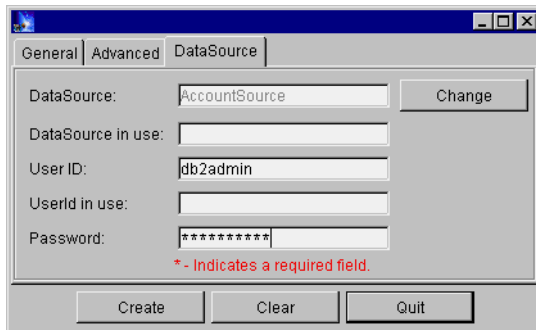


Figure 51. Specifying the data source for a container

- Click **Create** to create the EJB container.

4.3.5 Create the Account and Transfer EJBs in WebSphere

We will now add the Account and Transfer EJBs to the container.

- Right-click on the EJB container you just created and select **Create -> EnterpriseBean**.

A dialog box like Figure 52 on page 72 appears.

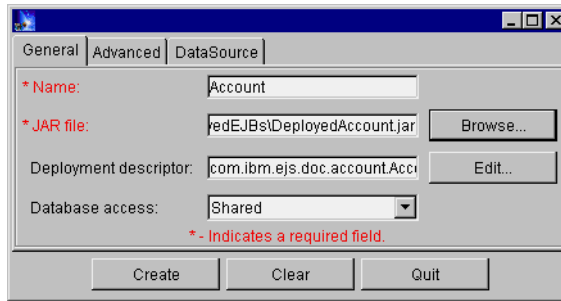


Figure 52. Creating an EJB in WebSphere

- Enter the name `Account` and click **Browse** to select the Java archive of the account example.

You find it in the `AppServer\deployableEJBs` subdirectory of your WebSphere directory. We picked:

`D:\WebSphere\AppServer\deployableEJBs\Account.jar`

If the dialog box shown in Figure 53 appears, click **Yes**.

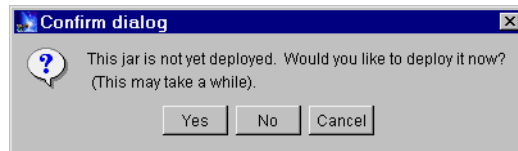


Figure 53. Deploying an EJB in WebSphere

You do not have to specify a data source since this was already done in the container. EJBs without data source use the data source of the container.

- Click **Create** to deploy the EJB.

Note that WebSphere has created a Java archive for the new bean in the `Appserver\deployedEJBs` subdirectory of your WebSphere directory. You need this archive when you want to access the EJB from an EJB client program. More information about the use of EJBs is in 5.3, “Calling Enterprise Java Beans that are managed by WebSphere” on page 118.

You can deploy multiple beans in one EJB container.

4.3.5.1 Create the Transfer EJB

Deploy the Transfer EJB in the same way as we just described for the Account EJB.

4.3.5.2 Start the deployed EJBs

After you have deployed the EJBs you start them. You start an EJB by highlighting its name and pressing the green button. If only the red button is visible the EJB is already running.

4.3.6 Deploying the servlets and JSPs

The servlets in the example must be compiled before you can deploy them. You use VA Java for this.

- In your VA Java Workbench, expand the project **IBM Account example** and select the package **com.ibm.ejs.doc.client**.
- Select **File -> export**; in the dialog box select **Directory**; click **Next >**.
- Select the servlet directory of your WebSphere server.

We selected:

D:\WebSphere\AppServer\servlets

Make sure that **.class** is selected as shown in Figure 54 on page 74.

- Click **Finish**.

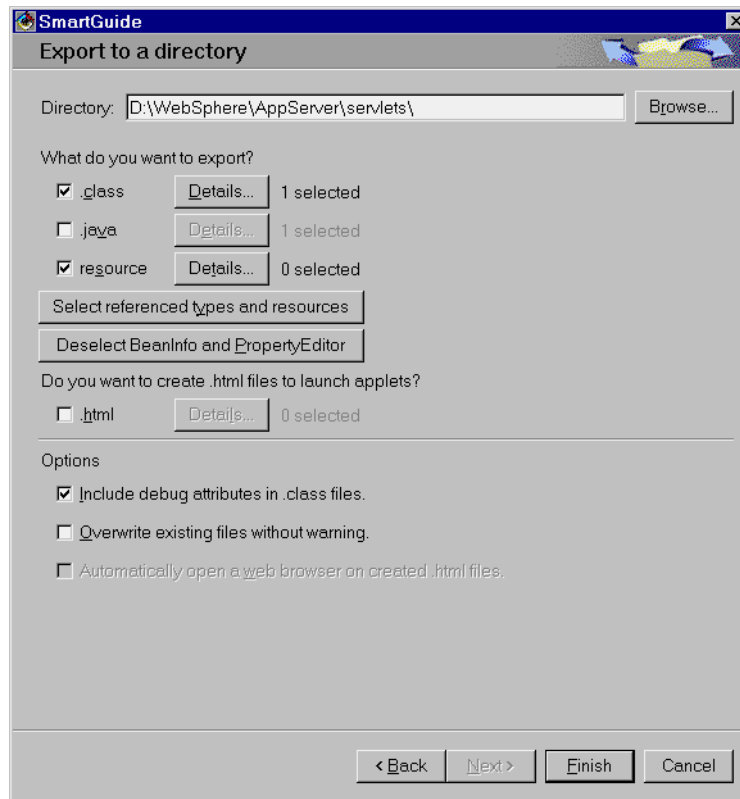


Figure 54. Exporting the account example servlets

The HTML files and JSPs that belong to the account example application are located in the AppServer\WebSphereSamples\EJBs subdirectory of your WebSphere directory. Copy the directories Account and Transfer, and all of their files, into the Domino HTML directory. For example, the target path for our Account HTML and JSP files were:

D:\Lotus\Domino\Data\domino\html\Account

Now the account example is deployed and you can test it using the URL:

`http://yourhostname/Account/Create.html`

where yourhostname is the name of your Web server, for example:

`gefion.lotus.com.`

Your browser should display the screen shown in Figure 55 on page 75.

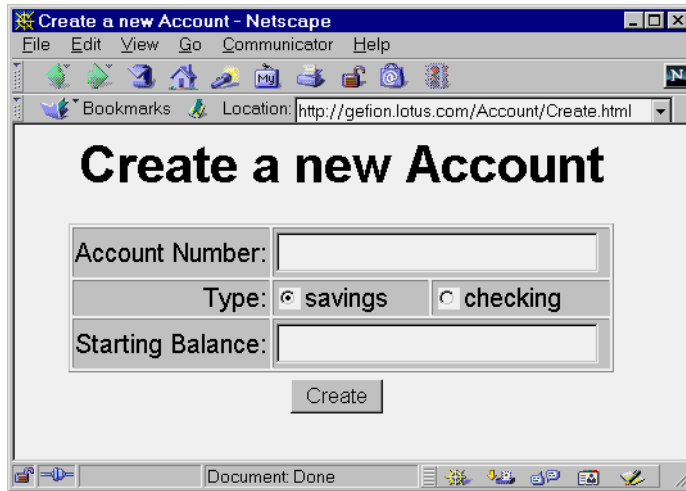


Figure 55. IBM Account example

Fill out the fields and create a test account. If the sample works correctly you will get an answer in your Web browser below the Create button, similar to this:

Created account: 24072000, Type: checking, Balance: 5000

We will modify the example later in this and the following chapter. However, first we will look into how to access Domino databases from WebSphere using JDBC.

4.4 Using JDBC to access Domino R5

We will start with the simplest example of application-level integration between Domino and WebSphere, where we simply access Domino databases like any other database through JDBC. In this section we will look at:

- How to install the JDBC driver for Domino
- Accessing Domino through JDBC from a servlet
- Accessing Domino through JDBC from a JavaServer Page (JSP)

4.4.0.1 Installing the JDBC driver for Domino

Lotus provides a JDBC driver for Domino R5. You can download it at:

<http://www.lotus.com/developers/devbase.nsf/homedata/homejdbc>

After you have downloaded the Domino R5 JDBC driver, install it on your WebSphere server. After the installation you find the Java archive *JdbcDomino.jar* and the DLLs that are used for the JDBC access in the subdirectory *lib* of the directory you installed the JDBC driver to.

You must add the *JdbcDomino.jar* file to your classpath or the directory containing the DLLs to your library paths to access Domino data from WebSphere via JDBC. See 4.2.1, “Modifying the path WebSphere uses” on page 66 for instructions on how to modify these paths in WebSphere.

We added the Java archive to the following class path variable in the *admin.config* file:

```
com.ibm.ejs.sm.adminserver.classpath.
```

Here is what we added:

```
d:/Lotus/Domino/JdbcSql/lib/JdbcDomino.jar
```

Note: If you installed a Notes client after you installed Domino on your server, JDBC will use the Notes ID you were using in this client to access Domino. Make sure that this ID is not password-protected because the servlets and JSPs do not respond if they encounter a password-protected ID.

Be aware that every user of your servlet accesses the Domino databases using the same Notes ID. You should use WebSphere security or build application-based security if you plan to access Domino R5 via JDBC.

You can try to run some of the simple examples supplied with the JDBC driver for Domino to verify that the JDBC and Domino combination works before adding WebSphere to the equation.

4.4.1 Creating a servlet that uses JDBC to access Domino R5 data

In this section we will explain how to write a servlet that reads Domino R5 data via JDBC. Our example will display all person entries in the Domino R5 directory.

4.4.1.1 Creating the servlet class

To create the servlet class start VA Java and do the following:

- In the VA Java Workbench expand the **IBM Account example** project and highlight the **com.ibm.ejs.doc.sg245955** package.
- Select **Selected -> Add -> Class**
 - Enter *ReadNames* as the class name.

- Enter or select **javax.servlet.http.HttpServlet** as the superclass.
- Click **Finish** to create the class.

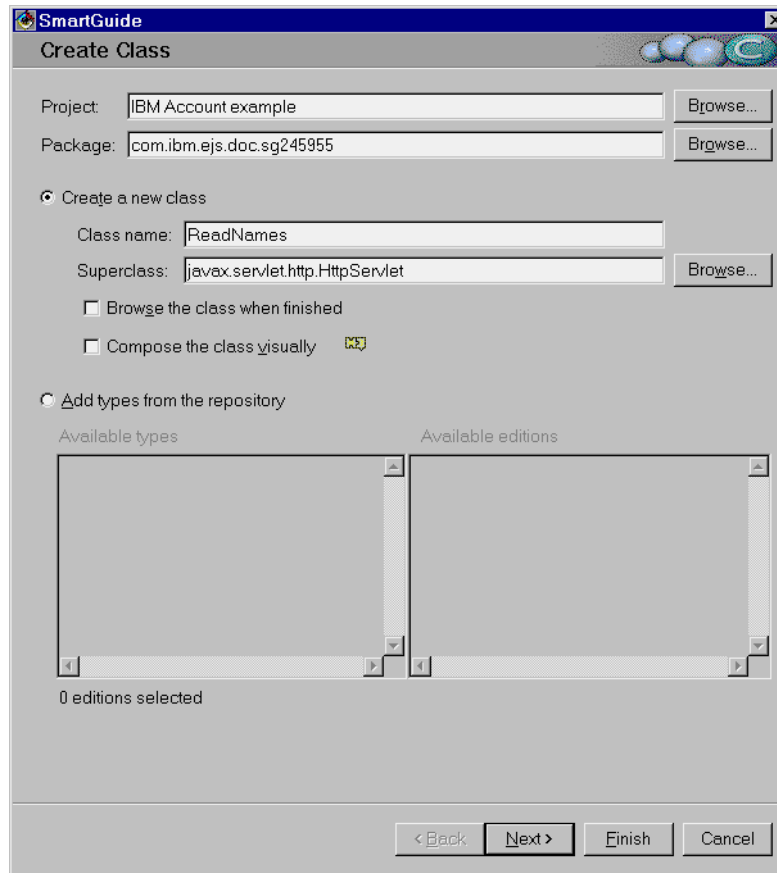


Figure 56. Creating a new class in VA Java

- Expand the **sg245955** package, select the **ReadNames** class and see the class definition in the source pane.

Before the class definition, insert the following import statements for all servlet classes and the Java JDBC classes into the new servlet:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
```

- Add a private variable to contain the JDBC connection to the database. Include the following code at class level:

```
private Connection objCon = null;
```

This connection will be created when the servlet is called for the first time in a session and is kept until the WebSphere servlet engine is stopped.

Your VA Java screen should look like Figure 57.

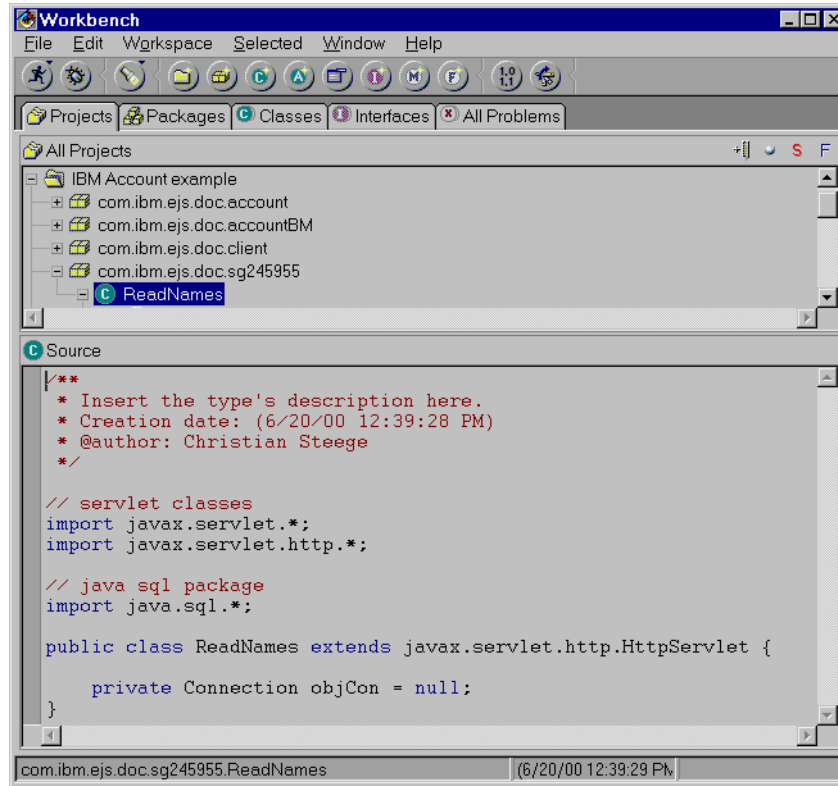


Figure 57. Definition of the ReadNames class

We will now add the init, destroy and doGet methods to the class.

4.4.1.2 The init and destroy methods of the servlet

If you create a method called *init* in a servlet, the container (in our case that means WebSphere) calls this method when the servlet is started. You can define the servlet to be started when WebSphere is started. Otherwise, the servlet is initialized the first time it is called using its URL. The init method has to throw a ServletException if it fails.

We will open the JDBC Connection in the init method.

You can create a new method in VA Java like this:

- Highlight the class you want to add the method to and select **Selected -> Add -> Method**.
- Specify the name of your new method and click **Finish**.

To create the init method you select the **ReadNames** class and specify the name as **public void init ()**.

- Then you select the **init** method and enter the following code:

```
public void init(ServletConfig objConfig) throws ServletException {
    super.init(objConfig);
    try {
        Class.forName ("lotus.jdbc.domino.DominoDriver");
        objCon = DriverManager.getConnection
            ("jdbc:domino:/names.nsf/yourservername");
    } catch (Exception e) {
        System.out.println (e.getMessage());
    }
}
```

In this code we connect to the Domino database we want to work with. Replace yourservername by the abbreviated Domino name of your Domino R5 server. We used gefion.

The *destroy* method is called just before the servlet is stopped. We will use it to close the JDBC connection.

- Create a destroy method in the same way you created the init method and enter the following code:

```
public void destroy() {
    try {
        objCon.close();
    } catch (Exception e) {
        System.out.println (e.getMessage());
    }
}
```

Finally, we will create the method that will do the major part of the work.

4.4.1.3 The doGet method of the servlet

When creating a servlet, VA Java creates a method called *service*. Although this method responds to all requests the servlet gets, it is better to distinguish between the different types of requests (GET or POST). This prevents errors when calling the servlet using an unexpected method.

Therefore we will rename the service method to doGet.

- Rename a method in VA Java by right-clicking on the method name and selecting **Reorganize -> Rename**.

Select the **service** method and rename it to **doGet**.

The doGet method is called when the servlet is called from a browser via its URL.

- In addition, change the type of the parameters from **ServletRequest** to **HttpServletRequest** and from **ServletResponse** to **HttpServletResponse**. These parameter types provide more information.

In the doGet method we will select all entries of the people view in the Domino R5 directory and display them in a HTML page. By modifying the SQL statement in the sample code, you could create or modify data too. When accessing Domino R5 via JDBC you can use all view names and all form names of the database you are using as SQL table names.

- Enter the following code in the doGet method:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws
    javax.servlet.ServletException, java.io.IOException {
    // begin html output
    res.setContentType("text/html");
    res.setHeader("Pragma", "no-cache");
    res.setHeader("Cache-control", "no-cache");
    java.io.PrintWriter pw = res.getWriter();
    pw.println("<HTML>");
    pw.println("<HEAD>");
    pw.println("<TITLE>Persons</TITLE>");
    pw.println("</HEAD>");
    pw.println("<BODY>");
    pw.println("<H1 ALIGN=\"CENTER\">Persons in Domino directory</H1>");
    pw.println("<TABLE>");
    // execute query
    try {
        Statement objGetNames = objCon.createStatement();
        ResultSet objResNames = objGetNames.executeQuery
            ("SELECT * FROM People");
        // print lines
        String sLine = "";
        while (objResNames.next()) {
            sLine = "<TR>";
            for (int iCol = 1;
                iCol <= objResNames.getMetaData().getColumnCount(); iCol++)
            {
                Object objCell = objResNames.getObject(iCol);
```

```

        if (objResNames.isNull()) {
            sLine = sLine + "<TD> </TD>";
        } else {
            sLine = sLine + "<TD>" + objCell.toString() + "</TD>";
        }
    }
    sLine = sLine + "</TR>";
    pw.println(sLine);
}
} catch (Exception e) {
    pw.println(e.getMessage());
    System.out.println(e.getMessage());
}
// end html output
pw.println("</TABLE>");
pw.println("</BODY>");
pw.println("</HTML>");
}

```

- Select the class definition and save the code.

If no error is reported, the servlet code is correct and you are ready to deploy the servlet on the WebSphere server.

4.4.1.4 Deploying and testing the ReadNames servlet in WebSphere

We will now deploy the servlet we just developed in VA Java.

- In VA Java highlight the **ReadNames** class and select **File -> Export**.
- Select **Directory** and click **Next >**.
Select the servlet directory of your WebSphere server. We used:
D:\WebSphere\AppServer\servlets
Make sure that **.class** is selected.
- Click **Finish** to deploy the ReadNames servlet.

The deployed servlet can now be tested using the URL:

<http://yourhostname/servlet/com.ibm.ejs.doc.sg245955.ReadNames>

where yourhostname is the host name of your server, for example *gefion.lotus.com*. The output from the servlet should look similar to Figure 58 on page 82.

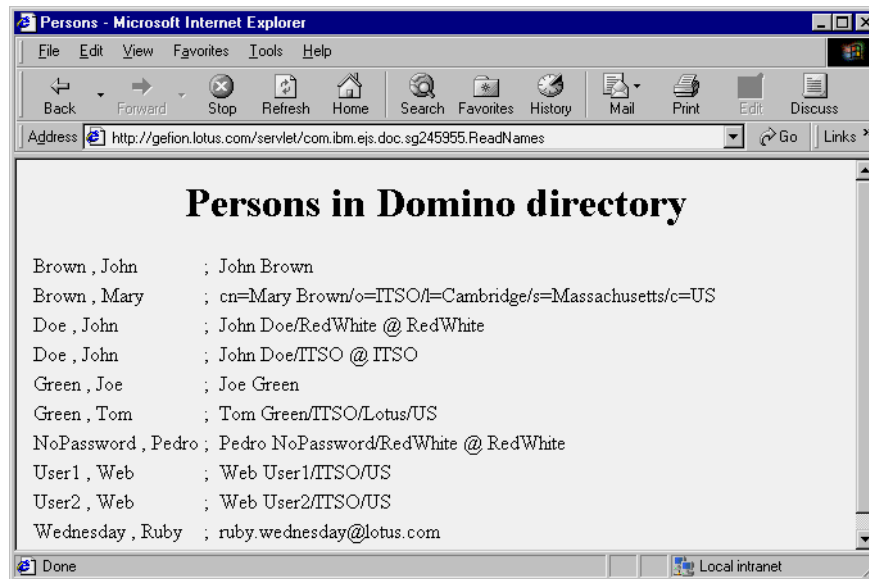


Figure 58. Output from ReadNames servlet

You can find more information about servlet URLs in 5.1.1, “Servlet URLs” on page 105.

4.4.2 Creating a JSP that uses JDBC to access Domino R5 data

The servlet we just created produces an HTML page by combining database output with HTML tags. JSPs are a different method to achieve the same result. However, the Java code in JSPs is harder to read and harder to debug. On the other hand, it is easier to write HTML code in JSPs because you can simply write the code as in HTML pages.

If your code contains complex logic and not much HTML you should create a servlet. If you want to display some dynamic values in a HTML page, in most cases a JSP is more appropriate.

You can use various tools (like IBM WebSphere Studio product or Netobjects Fusion ScriptBuilder) to create JSPs, or you can use a simple editor. For the samples in this Redbook a simple editor like the *notepad* in Windows NT is sufficient.

4.4.2.1 Creating the HTML page

The JSP we want to create is similar to the ReadNames servlet, but instead of hard coding the server name, database name, and view name we will let

the user specify these values dynamically. The JSP can thus read any Domino view.

The top of the JSP consists of a form that users can use to specify the view they want to see. When pressing the submit button the JSP will call itself to display the view below the entry fields.

- Create your JSP file in the Domino R5 HTML directory and call it ReadView.jsp. We used the following file name:

D:\Lotus\Domino\Data\domino\html\ReadView.jsp

- You can use any editor to create the HTML code. Enter the following HTML code:

```
<HTML>
<BODY>
<Form METHOD=POST ACTION="ReadView.jsp" Name=ShowViews>
<TABLE>
<TR><TD><B>Server:</B></TD><TD><INPUT NAME="server" VALUE=""
size=20></TD></TR>
<TR><TD><B>Database:</B></TD><TD><INPUT NAME="db" VALUE=""
size=20></TD></TR>
<TR><TD><B>View:</B></TD><TD><INPUT NAME="view" VALUE=""
size=20></TD></TR>
</TABLE>
<CENTER><INPUT TYPE=SUBMIT VALUE="Show view"></CENTER>
</Form>
<TABLE>
</TABLE>
</BODY>
</HTML>
```

There are two tables in the form. The second table will be used to display the view contents.

We now have a plain HTML form. In the next section we will add some logic to the form.

4.4.2.2 Requesting parameters

In JSPs you can use the `<%= ... %>` tag to calculate a single Java statement. Often this is used to display a parameter that can be read by using the request object. The request object contains any parameters that are passed as part of the URL or as part of a POSTed form.

We fill the fields in the form with the values of the last call using the *getParameter* method. The user can simply change one value (for example, the view) and leave the other values as they are.

- Include the tags highlighted in bold in your JSP.

The JSP should now contain the following code:

```
<HTML>
<BODY>
<Form METHOD=POST ACTION="ReadView.jsp" Name=ShowViews>
<TABLE>
<TR><TD><B>Server:</B></TD><TD><INPUT NAME="server" VALUE="
<%= request.getParameter ("server") %>" size=20></TD></TR>
<TR><TD><B>Database:</B></TD><TD><INPUT NAME="db" VALUE="
<%= request.getParameter ("db") %>" size=20></TD></TR>
<TR><TD><B>View:</B></TD><TD><INPUT NAME="view" VALUE="
<%= request.getParameter ("view") %>" size=20></TD></TR>
</TABLE>
<CENTER><INPUT TYPE=SUBMIT VALUE="Show view"></CENTER>
</Form>
<TABLE>
</TABLE>
</BODY>
</HTML>
```

We are not done yet. We keep the parameters the user keyed in and pass them back in the reply, but we also need some logic to do the actual database lookup.

4.4.2.3 Scriptlet code to display the view

Now we will include a chunk of Java code (called a scriptlet) to display the view into the second table. To be able to use JDBC we must include the Java JDBC classes. You do not need to include the servlet classes into JSP, because the JSP precompiler adds these classes automatically.

You cannot use the init and destroy methods in JSPs, so we must open and close the connection for each call. Apart from this, the logic for the display of the view remains the same as in a servlet.

- Add the code highlighted in bold to your JSP. The JSP now has the following code:

```
<HTML>
<%@ language="java" import="java.sql.*" %>
<BODY>
<Form METHOD=POST ACTION="ReadView.jsp" Name=ShowViews>
<TABLE>
<TR><TD><B>Server:</B></TD><TD><INPUT NAME="server" VALUE="
<%= request.getParameter ("server") %>" size=20></TD></TR>
<TR><TD><B>Database:</B></TD><TD><INPUT NAME="db" VALUE="
<%= request.getParameter ("db") %>" size=20></TD></TR>
```



```

<TR><TD><B>View:</B></TD><TD><INPUT NAME="view" VALUE="
<%= request.getParameter ("view") %>" size=20></TD></TR>
</TABLE>
<CENTER><INPUT TYPE=SUBMIT VALUE="Show view"></CENTER>
</Form>
<TABLE>
<%
    try {
        Class.forName ("lotus.jdbc.domino.DominoDriver");
        Connection objCon = DriverManager.getConnection("jdbc:domino:" +
            request.getParameter ("db") + "/" +
            request.getParameter ("server"));
        java.io.PrintWriter pw = response.getWriter();
        Statement objGetNames = objCon.createStatement();
        ResultSet objResNames = objGetNames.executeQuery ("SELECT * FROM "
+
            request.getParameter ("view"));
        // print lines
        String sLine = "";
        while (objResNames.next()) {
            sLine = "<TR>";
            for (int iCol = 1;
                iCol <= objResNames.getMetaData().getColumnCount(); iCol++)
            {
                Object objCell = objResNames.getObject(iCol);
                if (objResNames.isNull()) {
                    sLine = sLine + "<TD> </TD>";
                } else {
                    sLine = sLine + "<TD>" + objCell.toString() + "</TD>";
                }
            }
            sLine = sLine + "</TR>";
            pw.println(sLine);
        }
        objCon.close();
    } catch (Exception e) {
        System.out.println (e.getMessage());
    }
%>
</TABLE>
</BODY>
</HTML>

```

You can call the JSP using the URL:

<http://yourhostname/ReadView.jsp>

where yourhostname is the host name of you server, for example *gefion.lotus.com*.

Your output should look similar to Figure 59.

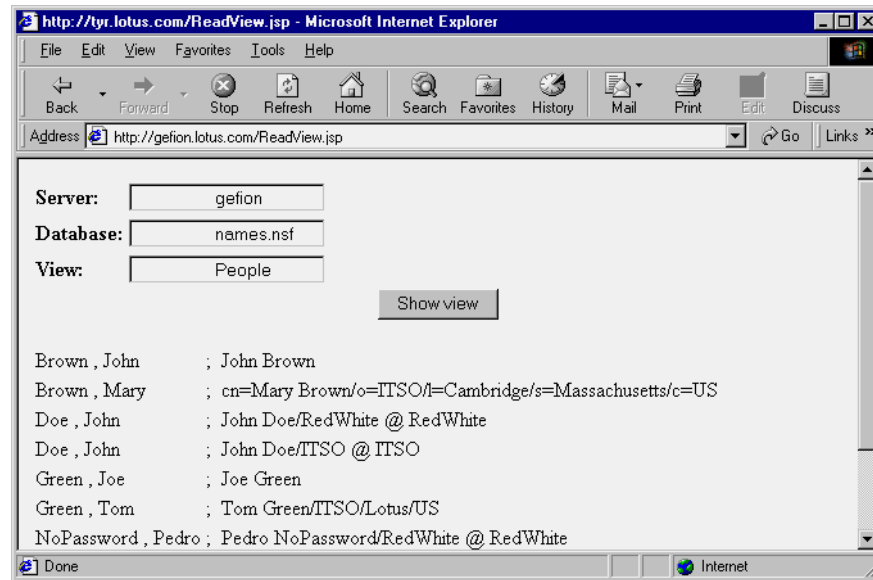


Figure 59. Output from ReadView.jsp

The names you enter into the fields are case sensitive.

Note: During the installation of WebSphere the JSP 0.91 support is installed by default, and we ran all of our JSP examples using this. However, you may want to use the JSP 1.0 support instead since it offers more functionality. The the level of JSP support in your WebSphere Web application is determined by which of the following two JSP support instances you have enabled using the WebSphere administrative console:

- `com.ibm.servlet.jsp.http.pagecompile.PageCompileServlet` mapped to `*.jsp` supports JSP 0.91.
- `com.sun.jsp.runtime.JspServlet` mapped to `application_Web_path/*.jsp` supports JSP 1.0.

This concludes our examples of accessing Domino using JDBC. We will now look at some examples where you not only access data but also use some of the functionality Domino provides.

4.5 Accessing Domino R5 using the Domino R5 Java classes

In many cases the preferred method of accessing Domino R5 data from servlets or JSPs will be using the Domino R5 Java classes.

These classes provide access to all Domino R5 features and provide more security options than JDBC, where all users are known to Domino using the same Notes ID. Using the Domino R5 Java classes, you can access remote Domino servers via CORBA/IIOP.

4.5.1 Creating a servlet that sends a Domino e-mail

In our first example of the use of the Domino classes we will create a servlet that sends an e-mail on the local server. This example requires the modifications to the WebSphere path settings that we described in 4.2.1, “Modifying the path WebSphere uses” on page 66.

We will create a new servlet as a copy of the *CreateAccount* servlet class that belongs to the IBM Account example. This servlet creates a new account using the account EJB. We will send an e-mail to a Domino recipient that confirms the account creation. The e-mail can also initiate a workflow, but we do not show this in our example.

4.5.1.1 Create a copy of the existing CreateAccount servlet

Start VA Java. Expand the **IBM Account example** project and the **com.ibm.ejs.doc.client** package.

- Right-click **CreateAccount** and select **Reorganize -> Copy**.

Enter or select the package **com.ibm.ejs.doc.sg245955** and check **Rename the copy** as you see in Figure 60.

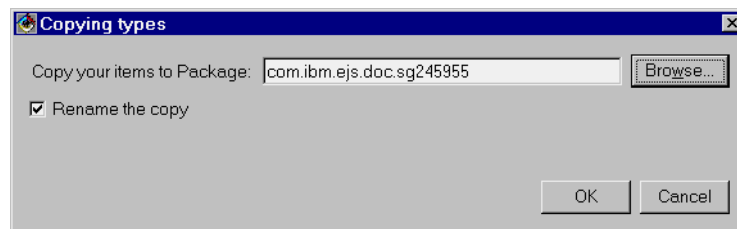


Figure 60. Copy a class in VA Java

- Click **OK** and enter the new name `CreateAccountMail`, then click **OK** again.

Because we will be using Domino R5 classes in our servlet, we include them at the top of the class definition code by adding:

```
import lotus.domino.*;
```

4.5.1.2 The sendMail method

We will now create a new method in the servlet, called *sendMail*. As parameters we pass a print writer that enables the method to print messages to the browser window. We will print the title of the database we open.

The sendMail method must return true if the mail could be sent and false if not.

- Make sure the **CreateAccountMail** class is selected in VA Java.
- Then select **Selected -> Add -> Method**.
 - Specify the name of your new method as `sendMail`.
 - Leave all other values as is, and click **Next >**.
- On the Attributes pane specify the return type as **boolean**.

Click **Add...** to add the following parameters.

Table 1. Parameters for sendMail method in CreateAccountMail class

Name	Reference Type
pw	PrintWriter
sRecipient	String
sSubject	String
sMessage	String

The Parameters dialog stays open until you click **Close**.

The create method dialog box should now look like Figure 61 on page 89. You can add parameters during the creation of a method by clicking the **Add...** button.

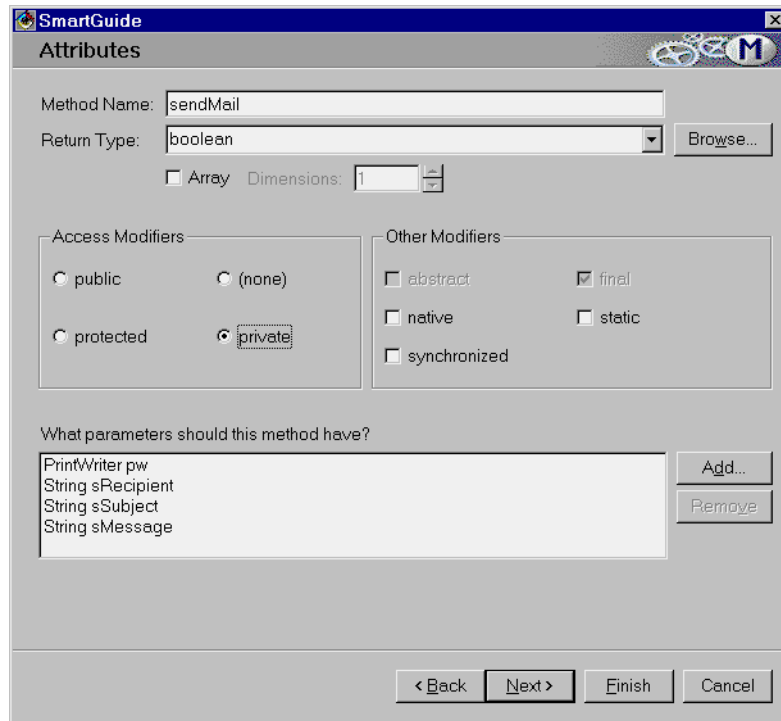


Figure 61. Creating a new method in VA Java

- Click **Finish** to have the method created.

In the `sendMail` method we must create a new Notes thread. This is important to prevent memory leaks. Then the Notes session can be created and used to access a database. We are creating the mail directly in the `mail.box` database. After the mail is sent we terminate the Notes thread.

- Enter the following code into the `sendMail` method:

```
private boolean sendMail(PrintWriter pw,String sRecipient,
    String sSubject,String sMessage) {
    boolean bSuccess = false;
    try {
        // initialize Notes tread and open notes session
        NotesThread.sinitThread();
        Session sesCurrent = NotesFactory.createSession();
        // Get database
        Database ndbMailbox = sesCurrent.getDatabase(null,"mail.box");
        if (!ndbMailbox.isOpen()) {
            ndbMailbox.open();
        }
    }
}
```

```

    }
    pw.println("<BR>Db is open: " + ndbMailbox.getTitle());
    // Create and send memo
    Document docMemo = ndbMailbox.createDocument();
    docMemo.appendItemValue("Form", "Memo");
    docMemo.appendItemValue("Subject", sSubject);
    RichTextItem rtBody = docMemo.createRichTextItem("Body");
    rtBody.appendText(sMessage);
    docMemo.send(sRecipient);
    bSuccess = true;
} catch (NotesException e) {
    pw.println("<BR>NOTES ERROR #" + e.id + " " + e.text);
} catch (Exception e) {
    pw.println("<BR>ERROR: " + e.getMessage());
    e.printStackTrace(pw);
} finally {
    NotesThread.sternThread();
    return bSuccess;
}
}

```

This code accesses the server with server rights and the author of the mail is the server. If you want to log in as a user, you replace the line in which the session is created by this:

```

Session sesCurrent = NotesFactory.createSession((String (null),
    "username", "password");

```

Username and password must be a valid Domino username and this users Internet password. Now the right of the specified user applies to the servlet actions and e-mails are sent with his identity.

4.5.1.3 Modifications to the doGet method

To send the email after the account is created we will call the *sendMail* method in the existing doGet method of the servlet.

- Add the code lines that are highlighted in bold to the end of the doGet method.

```

...
out.println("<td align=\"left\">" + messageLine + "</td>");
out.println("</tr>");
if (sendMail (out,"recipient","Account creation process",messageLine)) {
    out.println("<tr><td align=\"left\">Mail was sent</td></tr>");
    } else {
    out.println("<tr><td align=\"left\">Mail could not be " +
        " sent</td></tr>");
    }
}

```

```

out.println("</table>");
out.println("</form>");
...

```

Replace recipient with the user you want to send the mail to. For testing purposes you can use your own Domino mail account.

4.5.1.4 Deploying and testing the servlet

We will now export the servlet into the WebSphere servlet directory.

- In VA Java make sure the class **CreateAccountMail** is selected.
 - Select **File -> Export**
- Select **Directory** and specify the WebSphere servlet directory. We used:

D:\WebSphere\AppServer\servlet

- Click **Finish**.

Since you are accessing Domino from a Java program, make sure that the user you specified can access the server from Java, Javascript, or COM programs. This can be specified on the Security tab in the server document.

Whether you need restricted access or unrestricted access depends on the actions you want to perform within your servlet.

Create new databases:	Destinations allowed:
Create replica databases:	
Allowed to use monitors: *	
Not allowed to use monitors:	
Administer the server from a browser: Christian Steege/ITSO	
Agent Restrictions Who can -	Java/COM Restrictions Who can -
Run personal agents:	Run restricted Java/Javascript/COM: Access Server
Run restricted LotusScript/Java agents:	Run unrestricted Java/Javascript/COM:
Run unrestricted LotusScript/Java agents: freja/ITSO	

Figure 62. Security tab on server document

- Now you copy the file create.html that you find in the Account subdirectory of your Domino R5 directory. Name the copy **createwithmail.html**. Edit the new HTML file and change the form tag to:

```

<FORM METHOD="GET"
ACTION="/servlet/com.ibm.ejs.doc.sg245955.CreateAccountMail">

```

- Now call this URL in your Web browser:

http://yourhostname/Account/createwithmail

- Fill out the fields and click **Create**. The Account EJB creates the account and Domino sends the mail. You should get a message like the following back in your Web browser:

```
Db is open: Outgoing Mail
Created account: 23072000, Type: checking, Balance: 7700
Mail was sent
```

4.5.2 Accessing a remote server from a servlet using IIOP

If you want to access a remote Domino server via IIOP instead of a local server you must include the file *NCSOW.jar* into the WebSphere classpath. This file comes with Domino R5.04. The Domino server software can be an earlier version (we tested with R5.03).

Note: Do not include NCSO.jar into a WebSphere 3.x classpath. Even if you add NCSOW.jar, the IIOP connection is not possible until you remove NCSO.jar.

- Add NCSOW.jar to the WebSphere class path.

See 4.2.1, “Modifying the path WebSphere uses” on page 66 for information how to modify path entries in WebSphere. We added:

```
D:/Lotus/Domino/Data/domino/java/NCSOW.jar;
```

If you are not using the Domino R5 server as your HTTP server, an additional step is necessary before you can access Domino via IIOP. You must copy the file *diiop_ior.txt* to the HTML root directory of your Web server. You find this file in the *domino\html* subdirectory of your Domino data directory.

To create the mail on a remote server we will modify the *sendMail* method of the *CreateAccountMail* servlet. When you create the Notes session, specify:

- Which host name to use (in the form *gefion.lotus.com*)
- The user you want to use for the connection
- The password of the user

A second change is required when opening the *mail.box* database. If you do not specify the server, or if you omit the third parameter of the *getDatabase* method, the connection fails. The only server name that is accepted is the Notes server name of the server you created the session connection to.

After these changes your *sendMail* method contains the following code:

```
private boolean sendMail(PrintWriter pw,String sRecipient,
    String sSubject,String sMessage) {
    boolean bSuccess = false;
```



```

try {
    NotesThread.sinitThread();
    Session sesCurrent = NotesFactory.createSession("yourhostname",
        "username", "password");
    // Get database
    String sCurrentServer = sesCurrent.getServerName();
    Database ndbMailbox = sesCurrent.getDatabase(sCurrentServer,
        "mail.box", false);
    if (!ndbMailbox.isOpen()) {
        ndbMailbox.open();
    }
    pw.println ("<BR>Db is open: " + ndbMailbox.getTitle());
    // Create and send memo
    Document docMemo = ndbMailbox.createDocument();
    docMemo.appendItemValue("Form", "Memo");
    docMemo.appendItemValue("Subject", sSubject);
    RichTextItem rtBody = docMemo.createRichTextItem("Body");
    rtBody.appendText(sMessage);
    docMemo.send(sRecipient);
    bSuccess = true;
} catch (NotesException e) {
    pw.println ("<BR>NOTES ERROR #" + e.id + " " + e.text);
} catch (Exception e) {
    pw.println ("<BR>ERROR: " + e.getMessage());
    e.printStackTrace(pw);
} finally {
    NotesThread.stermThread();
    return bSuccess;
}
}

```

Now the servlet uses IIOP and does not require the Domino R5 server and the WebSphere server on the same machine.

You must export your servlet again, using the following steps:

- In VA Java make sure the class **CreateAccountMail** is selected.
- Select **File -> Export**
 Select **Directory** and specify the WebSphere servlet directory. We used:
 D:\WebSphere\AppServer\servlet
- Click **Finish**.

You will get a warning that the file already is in the directory. Accept to have it overwritten. Then try the new version of the servlet with the same URL as before in your Web browser:

http://yourhostname/Account/createwithmail

For a successful invocation of sendMail you will get a message sent to your Web browser similar to this (FREJA is the name of our remote Domino server):

Db is open: FREJA Mailbox

Created account: 222000, Type: savings, Balance: 67

Mail was sent

4.5.3 Accessing a Domino R5 server from a JSP

Access to Domino R5 from a JSP is very similar to access from a servlet, and it needs the same installation prerequisites. It works locally and via IIOP. The following code shows an example JSP that send a Domino e-mail via IIOP. The page is called SendMail.jsp.

```
<HTML>
<%@ language="java" import="lotus.domino.*" %>
<HEAD>
<TITLE>Send a Domino eMail</TITLE>
</HEAD>
<BODY BGCOLOR="FFFFFF">
<H1 ALIGN=CENTER>Send a Domino eMail</H1>

<FORM METHOD=POST ACTION="SendMail.jsp" NAME="SendMail">
<TABLE BORDER ALIGN=CENTER>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Username:</TD><TD><INPUT TYPE=TEXT
NAME="Username" SIZE="30" VALUE="<%= request.getParameter ("Username")
%>"></TD></TR>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Password:</TD><TD><INPUT
TYPE=PASSWORD NAME="Password" SIZE=30 VALUE="<%= request.getParameter
("Password") %>"></TD></TR>
<TR></TR>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Domino Server:</TD><TD><INPUT
TYPE=TEXT NAME="DominoServer" SIZE=30 VALUE="<%= request.getServerName()
%>"></TD></TR>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Recipient:</TD><TD><INPUT TYPE=TEXT
NAME="Recipient" SIZE=30></TD></TR>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Subject:</TD><TD><INPUT TYPE=TEXT
NAME="Subject" SIZE=30></TD></TR>
<TR BGCOLOR="#cccccc"><TD ALIGN=RIGHT>Message:</TD><TD><TEXTAREA
NAME="Body" ROWS=5 COLS=50></TEXTAREA></TD></TR>
</TABLE><BR>
<CENTER><INPUT TYPE="SUBMIT" NAME="Send" VALUE="Send"></CENTER>
</FORM>

<%
```

```

java.io.PrintWriter pw = response.getWriter();
try {
    NotesThread.sinitThread();
    String sCurrentServer = request.getParameter ("DominoServer");
    Session sesCurrent = NotesFactory.createSession(sCurrentServer,
        request.getParameter ("Username"),
        request.getParameter ("Password"));
    sCurrentServer = sesCurrent.getServerName();
    pw.println ("<BR>Notes session open on " + sCurrentServer);
    // Get database
    Database ndbMailbox = sesCurrent.getDatabase(sCurrentServer,
        "mail.box", false);
    if (!ndbMailbox.isOpen()) {
        ndbMailbox.open();
    }
    pw.println ("<BR>Db is open: " + ndbMailbox.getTitle());
    // Create and send memo
    Document docMemo = ndbMailbox.createDocument();
    docMemo.appendItemValue("Form", "Memo");
    docMemo.appendItemValue("Subject", request.getParameter ("Subject"));
    RichTextItem rtBody = docMemo.createRichTextItem("Body");
    rtBody.appendText(request.getParameter ("Body"));
    docMemo.send(request.getParameter ("Recipient"));
    pw.println ("<BR>Memo was sent");
} catch (NotesException e) {
    pw.println ("<BR>NOTES ERROR #" + e.id + " " + e.text);
} catch (Exception e) {
    pw.println ("<BR>ERROR: " + e.getMessage());
    e.printStackTrace(pw);
} finally {
    NotesThread.stermThread();
}
%>

</BODY>
</HTML>

```

If you put the page into your Domino HTML directory, you can call it using the URL:

<http://yourhostname/SendMail.jsp>

In your Web browser, the JSP will look like Figure 63 on page 96.

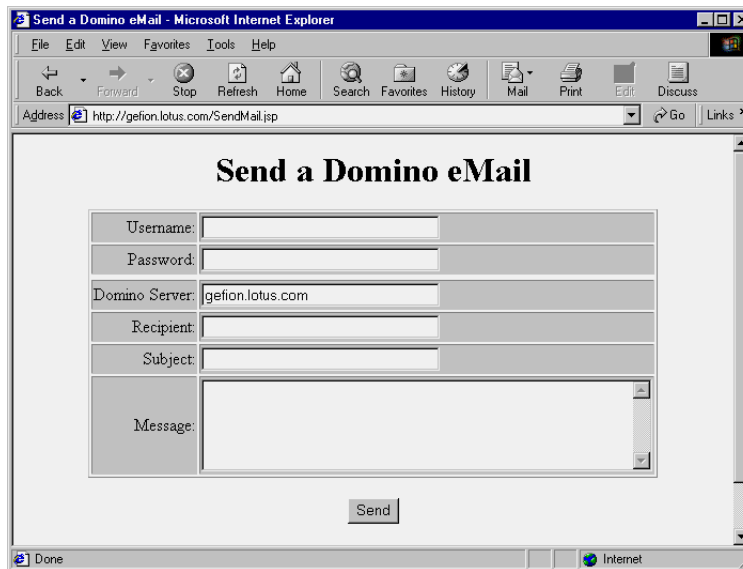


Figure 63. *SendMail.jsp*

Note: The JSP will attempt to send an email immediately when you load it the first time, so you will get a message below the Send button (not shown in the figure) indicating *No 'SendTo' field in document*. You can avoid this message by checking that the parameters are valid on your Java code in the JSP or you can have another HTML page invoke the JSP the first time around.

Fill out the fields and click **Send** to send an e-mail.

Upon a successful send a message similar to this will be returned in the JSP below the Send button:

```
Notes session open on CN=freja/O=ITSO
Db is open: FREJA Mailbox
Memo was sent
```

We have seen how to access Domino via IIOP from servlets and JSPs. We will now move on to an example of how to do it from an EJB.

4.6 Accessing Domino from EJBs

In this section we describe writing an EJB which uses the Domino Java API to access resources (in our example, to send mail) on a Domino server. Using an EJB to access Domino resources may be appropriate for an application if:

- The resources are accessed in the same way across different servlets or JSPs
- The access is part of a larger transaction
- The access involves capturing persistent data apart from Domino

Our discussion assumes basic familiarity with EJB concepts, such as EJB types (session vs. entity), components (bean class, home and remote or object interfaces) and deploying beans, especially using WebSphere. For a thorough discussion of these topics we suggest additional reading, such as the WebSphere product manual: *Writing Enterprise Beans in WebSphere*. Also see the discussion in 3.3, “Enterprise JavaBeans” on page 52.

You can write session beans that access Domino data. For entity beans, WebSphere does not support Container Managed Persistence using Domino databases as the persistent store. If you want to store entity beans in Domino R5, you have to use Bean Managed Persistence. Our example uses the Domino API in a session bean.

Note: In the examples so far in this chapter we have used VisualAge for Java Professional Edition. To develop EJBs you need the Enterprise edition of VisualAge Java. While we do discuss how we created an EJB that accesses Domino we do not give step-by-step instruction on how to do it. We give an overview of the process we went through and then discuss the Domino-specific details. If you want to learn about developing EJBs with VisualAge in general, refer to the IBM Redbook *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429.

4.6.1 Writing an EJB which uses the Domino Java API

We used the IBM VisualAge for Java (VA Java) Enterprise Edition 3.0 integrated development environment to build and deploy our EJB. VA Java creates much of the needed code and deployment information for an EJB automatically, allowing the developer to focus on the business logic. VA Java also contains a WebSphere test environment enabling EJBs to be tested and debugged prior to their deployment to a WebSphere server. Note that to use these EJB-related features, VA Java must run on a multi-processing operating system such as Windows/NT or AIX.

We added support for the Domino Java API to VA Java in the same way as described in 4.1, “Setting up IBM VisualAge for Java for the examples in this book” on page 59.

We were using the IIOP to access Domino. We built our EJB using the VA Java EJB creation wizard. Consult the VA Java help documentation or the

IBM Redbook *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429 for details on how to build an EJB.

You can get our sample EJB code in the RedDomTestEJB.jar file which is available at the Redbooks Web site. See Appendix C, "Additional Web material" on page 231 for more information.

In addition to the standard EJB methods, our EJB included the following Domino API code.

In the bean class *ejbCreate* method, we added code to call a method that create an API Session object to a remote Domino server using a preset Notes user name and password. We specified an existing Notes shortname and its Internet password. We specified the full TCP/IP host name of the Domino server. The Notes user name was part of a group which was given access to run unrestricted Java/JavaScript/COM agents on the Domino server. Figure 62 on page 91 shows where in the Domino server document these access rights are specified.

Note: It is possible for the Domino server DIIOP task to terminate the Notes session due to inactivity. In this case, your EJB code should always check the Notes session object to ensure it is still valid before performing the Domino functions. You can do this by attempting to invoke a Notes session method and, if a CORBA/IIOP exception is thrown, attempt to re-establish the Notes session.

Also in the bean class, we defined a data member called *itsNotesSession* to hold the API Session object so that subsequent EJB invocations could use the same Notes session and avoid the overhead of re-instantiating the Session object each time an instance of the EJB is created.

Here is the code for the *ejbCreate* and *createDomSession* methods:

```
public void ejbCreate() throws javax.ejb.CreateException,
java.rmi.RemoteException
{
    String aRetStr = createDomSession( "freja.lotus.com", "mbrown", "****" );
    if ( null != aRetStr )
        throw new javax.ejb.CreateException( aRetStr );
}

private String createDomSession( String theServer, String theUser, String
thePassword )
{
    String aRetStr = null;    // Assume success.
```

```

String aUser;
// Check if session already created and still valid.
if ( null != itsNotesSession )
{
    try {
        itsDomServer = itsNotesSession.getServerName();
        return aRetStr;
    } catch ( NotesException e ) {
        itsNotesSession = null;
    }
}
// Create a new Notes session.
// Note - if theServer is not null, NF uses IIOP to connect to host.
try {
    NotesFactory aNF = new NotesFactory();
    itsNotesSession = aNF.createSession( theServer, theUser,
        thePassword);
    aUser = itsNotesSession.getUserName();
    itsDomServer = itsNotesSession.getServerName();
    System.out.println( "Created remote session for user " + aUser +
        " on server " + itsDomServer );
} catch ( NotesException e ){
    aRetStr = "Notes Error #" + e.id + " " + e.text;
    e.printStackTrace();
    itsNotesSession = null;
} catch ( Exception e ) {
    aRetStr = "Exception: " + e;
    e.printStackTrace();
    itsNotesSession = null;
}
return aRetStr;
}
}

```

We also added a method named *sendMail* to send an e-mail message via the Domino server. This method was defined in the bean class and promoted (added) to the remote interface. Here is the code for the *sendMail* method:

```

public boolean sendMail(String theSender, String theRecipient,
String theSubject, String theMsg)
{
    boolean aRC = false;
    try {
        // Get Notes mail database
        Database ndbMailbox = itsNotesSession.getDatabase( itsDomServer,
            "mail.box", false );
        if ( !ndbMailbox.isOpen() )
            ndbMailbox.open();
    }
}

```

```

        // Create and send memo
        Document docMemo = ndbMailbox.createDocument();
        docMemo.appendItemValue( "Form", "Memo" );
        docMemo.appendItemValue( "Subject", theSubject );
        RichTextItem rtBody = docMemo.createRichTextItem( "Body" );
        rtBody.appendText( theMsg );
        docMemo.send( theRecipient );
        aRC = true;
    } catch ( NotesException e ) {
        System.out.println( "DomTest.sendMail: NOTES ERROR #" + e.id + " " +
            e.text );
    } catch ( Exception e ) {
        System.out.println( "DomTest.sendMail: ERROR: " + e.getMessage() );
        e.printStackTrace();
    }
    return aRC;
}

```

4.6.2 Deploying the EJB

After constructing the EJB, we used VA Java to generate an EJB jar file containing the components necessary to deploy the EJB. To generate an EJB jar file, highlight the EJB group or individual EJB, right-click to display the pop-up menu, then select **Export -> EJB JAR**. We exported the deployable EJB jar file to the WebSphere deployable EJBs directory. In our case the path was:

```
D:\WebSphere\AppServer\deployableEJBs
```

Note that you must use the Export EJB JAR function so that the jar file includes the necessary descriptor file which specifies how the EJBs are deployed.

Before we could deploy the EJB to WebSphere, we needed to make sure that the NCSOW.jar file was added to the class path used by WebSphere application server. See 4.5.2, “Accessing a remote server from a servlet using IIOP” on page 92 for more information about NCSOW.jar.

We deployed the EJB to the Default Container node. To do this, we highlighted this node in the tree shown in the Topology tab view of the WebSphere administrative console, right-clicked it to display the pop-up menu and selected **Create->EnterpriseBean**. As for any EJB, you can fill in the properties dialog directly from the deployable jar file by clicking **Browse**, then double-clicking on the deployable jar file (in the deployableEJBs subdirectory) and double-clicking on the descriptor (ser) file. Clicking **Create** in the dialog completes the deployment.

If you are running WebSphere with security enabled, before you can test access to the EJB, you must specify who is allowed to access the EJB. This is because all EJBs are protected by WebSphere security by default. Otherwise, the access attempt will fail with an authorization exception. To configure security for the EJB, you must:

1. Create (or choose an existing) enterprise application (EA) and add the EJB to it.
2. Configure the EJB resource for security (add method groups to its methods).
3. Assign permissions to the EA/method groups.

We permitted Everyone to access all method groups for the EA we created containing our Domino EJB.

4.6.2.1 Creating a servlet to invoke the EJB

We created a servlet to invoke our EJB. We copied the CreateAccountMail servlet which is part of the com.ibm.ejs.doc.sg245955 package (see 4.5.1, “Creating a servlet that sends a Domino e-mail” on page 87) to CreateAccountMailEJB.

Our modifications are shown in the following code. We added imports for our EJB to the class definition and a variable to hold the home object. We modified the init method to look up and obtain the home object for our EJB and modified the sendMail method to create the DominoTest EJB and invoke its sendMail method.

We added the import lines in bold to the CreateAccountMailEJB class:

```
// EJBs specific for this servlet.  
import com.ibm.ejs.doc.account.AccountHome;  
import com.ibm.ejs.doc.account.AccountKey;  
import com.ibm.ejs.doc.account.Account;  
  
import com.ibm.redbook.ejb.demo.DominoTestHome;  
import com.ibm.redbook.ejb.demo.DominoTest;
```

We also added the variable to hold the home object (in bold) to our class:

```
// EJB access.  
private AccountHome accountHome = null;  
private com.ibm.redbook.ejb.demo.DominoTestHome dominoTestHome = null;
```

We added the following lines to the init method right after where we get the home object for the Account EJB:

```
// Get the DominoTest EJB home object.  
homeObject = ctx.lookup( "DominoTest" );  
  
dominoTestHome = (DominoTestHome) javax.rmi.PortableRemoteObject.narrow(
```

```
(org.omg.CORBA.Object)homeObject, DominoTestHome.class );
```

Our sendMail method looks like this:

```
private boolean sendMail( PrintWriter pw, String sRecipient, String sSubject, String
sMessage )
{
    boolean bSuccess = false;
    DominoTest aDominoTestEJB = null;

    try {
        aDominoTestEJB = dominoTestHome.create();
        bSuccess = aDominoTestEJB.sendMail( "The Bank", sRecipient, sSubject, sMessage );
        if ( ! bSuccess )
            pw.println( "<BR>Error sending mail!" );
    }
    catch ( Exception e )
    {
        System.out.println( "dominoTest.sendMail: Exception: " + e.getMessage() );
        pw.println( "<BR>Exception: "+ e.getMessage() );
        e.printStackTrace();
    }

    return bSuccess;
}
```

Finally, we copied the createwithmail.html file to createwithmailEJB.html and changed it to invoke the CreateAccountMailEJB servlet.

4.6.3 Redeploying the EJB

We found that it is possible to redeploy the EJB after making code changes to it without having to recreate the EJB in WebSphere. Here are the steps we followed:

1. In VA Java, generate a deployed EJB jar file by highlighting the EJB or EJB group in the Enterprise Beans window pane of the EJB view, right-clicking it to display the pop-up menu, then selecting **Generate -> Deployed Code**.
2. Shut down the WebSphere application server which is running the EJB. In our case, this was the Default Server. We did this to ensure the EJB container was stopped and the EJB unloaded, and also to unload the servlet using the EJB.
3. From VA Java, export the deployed jar file by highlighting the EJB or EJB group in the Enterprise Beans window pane of the EJB view, right-clicking it to display the pop-up menu, then selecting **Export -> Deployed JAR**. You should overwrite the original deployed jar file known to WebSphere (usually in \WebSphere\AppServer\deployedEJBs).
4. Finally, restart the application server and the EA.

Note: If you have WebSphere security enabled *and* you have added or removed methods to the EJB's home interface, you will need to recreate the EJB since you must reconfigure security for the new methods. However, you can recreate the EJB from the deployed EJB jar file.

4.6.4 Summary

In this chapter we have covered how to set up VisualAge for Java to work with JSPs and servlets in WebSphere *and* to access Domino. We installed the WebSphere example that includes the Account and Transfer EJBs. Then we looked at how to access Domino from WebSphere. We started with common database access using JDBC from servlets and JSPs. Then we moved on to access the Domino Object Model (sending mail) from JSPs and servlets and finally we discussed how to access Domino from an EJB.

Chapter 5. Using WebSphere from Domino R5

In Chapter 1, “The platform for collaborative commerce” on page 1 we discussed the advantages of enhancing your Domino R5 applications with servlets, JavaServer Pages (JSPs) and Enterprise Java Beans (EJBs).

In this chapter, we describe several approaches to achieve this goal. There are two basic scenarios of using WebSphere in any Web application. You can access programs that are managed by WebSphere via:

- HTTP URLs from a Web browser or similar
- Java programs/agents

Both ways are possible using Domino R5 design elements. In this chapter we discuss the following:

- How to invoke servlets and how to set up Domino and WebSphere for servlet support
- How to invoke an existing servlet from a Domino form
- How to pass and return parameters from a JavaServer Page (JSP)
- Calling Enterprise Java Beans (EJBs) in WebSphere
 - on the same machine from a Domino agent
 - remotely via RMI from a Domino agent
 - using a servlet as intermediary

5.1 Invoking servlets from Domino R5

Servlets can be invoked from a browser using a URL. You can call a servlet from a Domino R5 application in all design elements that can call a URL; for example, in frame sets, pages, forms, or agents.

5.1.1 Servlet URLs

If WebSphere is installed and uses the Domino R5 HTTP service, Domino R5 sends all URLs that point to the servlet directory or to one of its subdirectories to WebSphere. The Domino R5 servlet directory is specified in the server document, as shown in Figure 64 on page 106.

[cs](#)
[Security](#)
[Ports](#)
[Server Tasks](#)
[Internet Protocols](#)
[MTAs](#)
[Miscellaneous](#)

[HTTP](#)
[Domino Web Engine](#)
[IIOP](#)
[LDAP](#)
[NNTP](#)

HTTP Sessions

Session authentication:

Disabled

Idle session timeout:

30 minutes

Maximum active sessions:

1000

Generating References to this Server

Does this server use IIS?

No

Protocol:

http

Host name:

Port number:

80

Java Servlets

Java servlet support:

None

Servlet URL path:

/servlet

Class path:

domino\servlet

Servlet file extensions:

Session state tracking:

Enabled

Idle session timeout:

30 minutes

Maximum active sessions:

1000

Session persistence:

Disabled

Figure 64. Internet Protocols/Domino Web Engine tab of the server document

Domino R5 internally replaces its servlet directory with the servlet directory of WebSphere. That means the servlets have to be in the WebSphere servlet directory or one of its subdirectories.

On the WebSphere side there is a corresponding path setting for the default invocation of servlets using their class name. This is the path of the invoker servlet in the default_app Web application. When you run a servlet that you have not explicitly configured in the administrative domain, the WebSphere Invoker servlet is called to support the servlet.

You can see how the default servlet path in WebSphere is defined in the administrative console in Figure 65 on page 107.

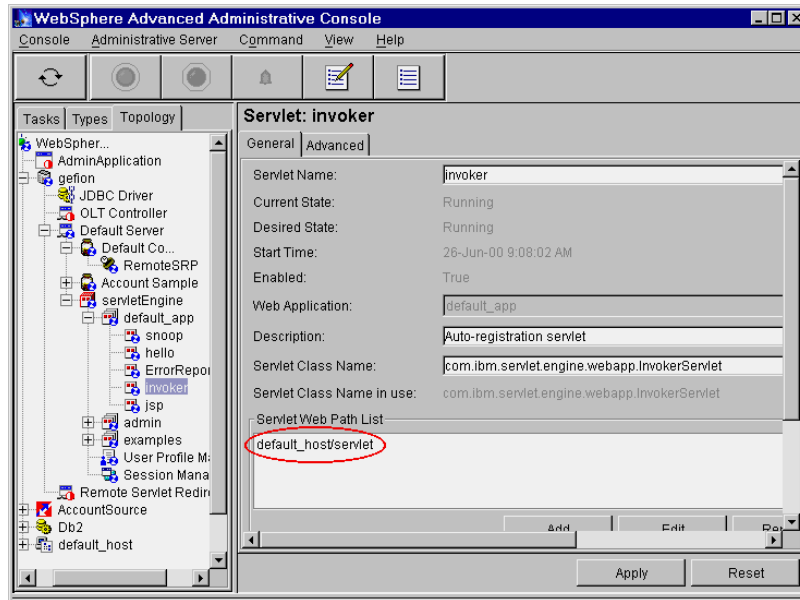


Figure 65. Servlet path in the WebSphere administrative console

If you want to change the default path for servlets you have to change the setting in the Domino server document, as well as the setting for the invoker servlet in default_app in the WebSphere administrative console.

However, you should only use the default path for servlets when you are experimenting and exploring; in general there should be no need to change the servlet path. In any production application you would organize your servlets in what is called an *enterprise application* in WebSphere. We will explain a bit about the difference between the default invocation of a servlet and a servlet that is defined as part of an enterprise application in WebSphere.

5.1.1.1 Default invocation of servlets

To run a servlet under WebSphere you simply need to deploy it to the servlet directory or one of its subdirectories. You do not need to do any configuration via the WebSphere administrative console: WebSphere calls the servlet if the URL points to a valid servlet class. Servlet URLs are case-sensitive and have to be spelled exactly like the class that is used to create the object in the servlet.

If the servlet is in a subdirectory of the WebSphere servlet directory, the directories have to be separated by a period (.). For example, if the Domino

R5 servlet directory is /servlet and you want to call a servlet that is in the com/ibm/ejs/doc/client subdirectory of the WebSphere servlet directory and that contains a class called CreateAccount, the URL to call this servlet would be:

```
http://yourhostname/servlet/com.ibm.ejs.doc.client.CreateAccount
```

After the servlet has performed its task, it can send back the code to display the next page or it can redirect to another URL.

5.1.1.2 Using another location for servlets and assigning aliases

If you do not want to put all your servlets into the standard servlet directory of WebSphere, you must create an enterprise application using the administrative console in WebSphere. For each application you can specify the Web application path that is used by the user to call resources of the application. In addition, you can specify one or more aliases for each servlet. These can be used by the user to call the servlet instead of the servlet class name. An example of this is the snoop servlet that we called in 2.7.4.1, “Verifying servlet support” on page 42. The name of the class file for the servlet is *SnoopServlet.class*. However, the servlet has been defined as being part of the default_app application with an alias of *snoop* and can thus be called using that alias instead of its class name.

5.1.2 Passing data to servlets in the URL

The easiest way to pass data to servlets is to create parameters in the URL that calls the servlet. The parameters are separated from the URL by a question mark (?), and the separator between parameters is an ampersand (&). The URL looks like:

```
http://yourhostname/servlet/com.ibm.ejs.doc.client.CreateAccount  
?account=7777&type=1&balance=4500
```

If the URL is called from a Domino R5 Form or a Domino R5 Agent, the data that is passed to the servlet can be collected from Domino R5 fields.

To test this, create an empty Domino Database. We called our database RedBanking, with a file name of banking.nsf. In this database you create a

form named Account and add a table to it. In this table create four fields, as shown in Table 2.

Table 2. Visible fields in the example Domino form

Field name	Type	Formula
UserName	Text, Computed	@Name ([CN] ; @UserName)
Account	Text, Editable	
Type	Radio Button, Editable	Radio button choices: Savings 1 Checking 2
Balance	Number, Editable	

To submit the form, create a button with the following JavaScript code in the **onClick** event:

```
document.forms[0].submit();
```

To pass the data to a servlet instead of creating a Notes document, the following three hidden fields are necessary:

Table 3. Hidden fields in the example Domino form

Field name	Type	Formula
Server_Name	Text, Editable	
SaveOptions	Text, Computed	"0"
\$\$Return	Text, Computed	"[http://" + Server_Name + "/servlet/com.ibm.ejs.doc.client .CreateAccount" + "?account=" + Account + "&type=" + Type + "&balance=" + @Text (Balance) + "]"

In the Domino R5 Designer the form now looks like that shown in Figure 66 on page 110.

User:	<input type="text" value="UserName T"/>
Account:	<input type="text" value="Account T"/>
Type:	<input type="radio"/> Type
Balance:	<input type="text" value="Balance #"/>
<input type="button" value="Create"/>	

Hidden fields:	
Server_Name (CGI Variable)	<input type="text" value="Server_Name T"/>
SaveOptions	<input type="text" value="SaveOptions T"/>
\$\$Return	<input type="text" value="\$\$Return T"/>

Figure 66. Example Domino Form in the Domino R5 Designer

If you submit the page, the servlet is called with the specified parameters.

In the servlet class, the URL parameters are collected from the `HttpServletRequest` parameter of the `doGet` method. The URL parameter names are case-sensitive and the URL must contain all parameters you try to retrieve. The `doGet` method of the `CreateAccount` sample contains the following code to retrieve the parameters:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
{
    ...
    // Read input parameters from URL.
    String[] accountArray = req.getParameterValues("account");
    String[] typeArray = req.getParameterValues("type");
    String[] balanceArray = req.getParameterValues("balance");
    ...
}
```

However, there are two things to consider when passing data as part of URLs:

- The first is that URLs may not contain spaces or certain special characters. This requires the replacement of some characters and all spaces. When doing this, select the replacement carefully; changing the replacement back to the original characters, which must be performed in the servlet, is only possible if the original string does not contain the replacement.
- The second issue with passing data in URLs is the limited length of the URL string. This limits the total amount of data that can be submitted in

one URL. For example, Netscape 4.04 allows about 2000 characters in one URL. This is dependent on the browser you are using and its version. If the URL is too long, most browsers simply truncate it without reporting an error.

5.1.3 Posting data to servlets from Domino R5 forms

You can use Domino R5 forms to collect data you want to post to servlets. This enables you to use most of the features of Domino R5 and the Domino R5 designer in your Web application forms if you have decided to use a data store other than Domino R5.

This technique makes it very easy to build a consistent user interface for combined Domino R5 and WebSphere applications. In addition, it helps you when you migrate parts of a Domino R5 application to WebSphere to achieve better performance.

5.1.3.1 Modifications to the account Domino form

When you open a Domino R5 form using a Web browser, Domino R5 automatically creates an HTML form tag. This tag includes the HTTP Method POST and an HTTP action to create a document in a Domino database when the form is submitted. Here is an example of a form tag created by Domino:

```
<FORM METHOD=post ACTION="/mydb.nsf/Account?OpenForm&Seq=1" NAME="_Account">
```

Since it is possible to have multiple HTML forms in one HTML page, you can simply close the HTML form tag that Domino R5 creates and start a new form. This form must have the method POST and the action must be the URL of the servlet you want to call.

In the form you created in 5.1.2, "Passing data to servlets in the URL" on page 108, add this pass-through HTML code at the top of the Domino form, above the first table:

```
</Form>
<FORM METHOD=POST
ACTION="/servlet/com.ibm.ejs.doc.sg245955.CreateAccountPost"
NAME="_AccountServlet">
```

In addition, modify the formula of the **Create** button to:

```
document.forms[1].submit()
```

Since the document contains two HTML forms, you have to use forms[1] instead of forms[0] to refer to the form you just defined. Alternatively, you can use the name (_AccountServlet) that you specified in the form HTML tag.

The hidden fields SaveOptions and \$\$Return are not necessary when you are posting data to a servlet. You can delete them. To enable the servlet to call the form again, add a new field to the Domino form:

Table 4. Hidden fields in the Domino form that posts data

Field name	Type	Default value formula
CurrentDatabase	Text, Editable	@ReplaceSubstring ("http://" + Server_Name + "/" + @ReplaceSubstring (@Subset (@DbName; -1); "\\\"; "/"); " "; "+")

If you select to hide this field for Web browsers in Domino Designer it will not be passed to the servlet. Instead, to prevent display of the field for the user, create a TYPE=HIDDEN tag in the HTML tab of the Field properties, as shown in Figure 67.

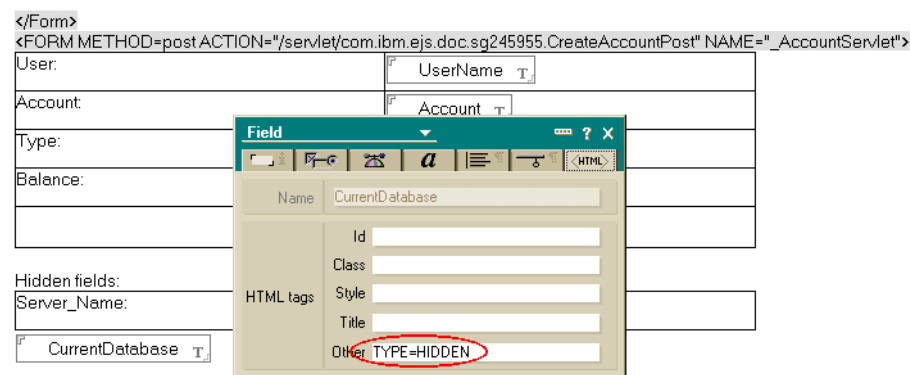


Figure 67. Domino R5 form that posts data to a servlet

5.1.3.2 Creating a servlet that accepts posted data

When calling a servlet with the HTML POST method, the doPost Method of the servlet is invoked. It has the same parameters as the doGet Method and you can retrieve each field of your HTML form as a parameter. The field names you use in the servlet are case-sensitive and the HTML form must contain all fields you try to retrieve.

We will now create a servlet that accepts posted data from our Domino form.

To do this you must have prepared IBM VisualAge for Java (VA Java) as described in 4.1, "Setting up IBM VisualAge for Java for the examples in this book" on page 59.

Then you do the following in VA Java:

- Expand the **IBM Account example** project and the **com.ibm.ejs.doc.client** package.
- Create a new copy of the CreateAccount class like this:
 - Right-click the class name **CreateAccount** and select **Reorganize -> Copy**.
 - VA Java asks where to copy the class to. Browse and pick (or type in) the package named:
`com.ibm.ejs.doc.sg245955`
 - Select the check box named **Rename the copy**, then click **OK**.
 - In the second Dialog box enter the new class name:
`CreateAccountPost`
and click **OK** again.
- Expand the **com.ibm.ejs.doc.sg245955** package and the methods below the new class **CreateAccountPost**.
- Rename the doGet method to doPost like this:
 - Right-click on the method name **doGet** and select **Reorganize - Rename**.
 - VA Java asks for the new name of the method. Type:
`doPost`.
and click **OK** again.

Our new servlet now responds to the HTML POST method instead of the HTML GET method.

We must now make a small change to get the parameters from the Domino form. Because the parameter names are case-sensitive, you must spell them exactly as in the Domino form. Change the three lines that request the parameters in the doPost method to:

```
...
// Read input parameters from HTML Form.
String[] asAccount = req.getParameterValues("Account");
String[] asType = req.getParameterValues("Type");
String[] asBalance = req.getParameterValues("Balance");
...
```

The HTML response is created at the end of the method. Here you add the HTML head tag that redirects the page to the Domino form if no error has occurred.

```
...
// --- Prepare and send HTML response. ---
res.setContentType("text/html");
res.setHeader("Pragma", "no-cache");
res.setHeader("Cache-control", "no-cache");
PrintWriter out = res.getWriter();

out.println("<html>");
out.println("<head>");
out.println("<title>" + createTitle + "</title>");
if (inputFlag && createFlag) {
    // Get contents of the CurrentDatabase field from the form
    String asDbURL[] = req.getParameterValues ("CurrentDatabase");
    // Redirect output to the Domino form
    out.println ("<meta http-equiv=\"refresh\" content=\"0; URL=\""
        + asDbURL[0] + "/Account?OpenForm\">");
}
out.println("</head>");
...
```

Another way to specify which form to use to show the result is to use the `sendRedirect` method of the `HttpServletResponse`. However, due to a bug in the `domino5.dll` in the version of WebSphere we were working with, invoking the `sendRedirect` method after the servlet had performed actions would throw an `IllegalStateException`. The workaround we apply here using the refresh meta tag will briefly show the form being returned from the servlet before redirecting to and showing the Domino form.

Since the redirection replaces the HTML page the servlet creates, you can delete most of the lines that create the body of this page; but do not delete the display of the message line because the redirection takes place only when the account creation is successful, and the message line contains possible error messages.

Note that in our example the message line is not transferred into the Domino form. You could achieve this by adding the message as a parameter to the URL you are redirecting the servlet response to. In Domino forms, parameters can be read by including a field named `Query_String` in the Notes form.

5.1.3.3 Deploying and testing the CreateAccountPost Servlet

Now we will export the servlet from VA Java into the WebSphere servlet directory so we can test it.

- In VA Java highlight the **CreateAccountPost** class and select **File -> Export**.

VA Java shows an export panel.

- Select **Directory** in the export panel and click **Next >**.
- Enter or select the WebSphere servlet directory as the target directory. We used:

D:\WebSphere\AppServer\servlets\

- Press **Finish** to export the servlet.

Now you can test the form by calling the URL:

`http://yourhostname/yourdbname.nsf/Account?OpenForm`

Fill out the fields and click the button. The servlet is called; it creates an account with the specified data and returns the user to a new blank account form served by Domino.

Note: This example is just an illustration of how to go from a Domino form to invoke a servlet and then redirect back to a new blank Domino form. In a real-life scenario the form shown to the user after the form is submitted should show a message confirming that the account is created and then let the user pick their next action.

5.2 Calling JavaServer Pages from Domino R5

JavaServer Pages (JSPs) are explained in 3.2, “JavaServer Pages” on page 50. Similar to servlets, JSPs can be called using special URLs. Every Domino R5 Design element that can call a URL can call a JSP.

5.2.1 JavaServer Page URLs

JSPs are stored in the default document path of your application. During the installation, WebSphere reads the Domino R5 path for Web resources and sets the document root of the default application to this path. You can modify it in the WebSphere administrative console if you want to put your JSPs into a different directory, as shown in Figure 68 on page 116. In addition, you can create new applications with different document roots.

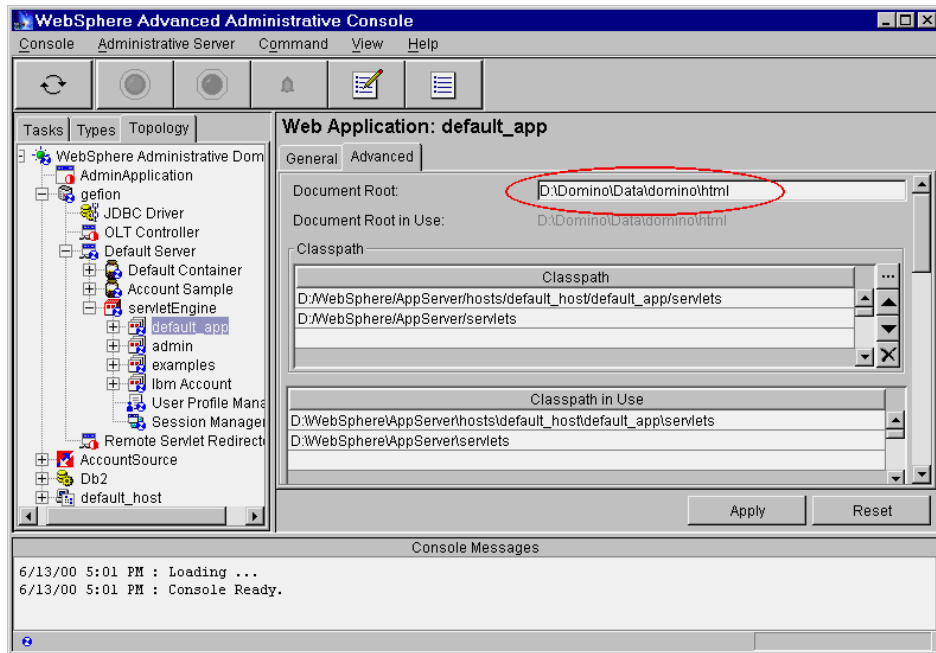


Figure 68. Modifying the document root of the default application in the administrative console

The URL of a JSP is case-sensitive, as is the URL of a servlet. If it is spelled incorrectly, you do not get any error from Domino R5, but the servlet that WebSphere creates from the JSP cannot be loaded and a long error page will be returned.

5.2.2 Passing data to JavaServer Pages

You can pass parameters to JSPs in the same way you pass them to servlets. JSPs do not distinguish between parameters of the URL and posted fields. That means you do not need to change the code of a JSP if you change the HTTP method (GET or POST) you use to call it.

The Java code within the JSP can access an object called *request*. This object contains all parameters of the URL or all fields of the form. You can extract the parameters with the *getParameter* method. The parameters differ from the servlet parameters, since they are strings instead of string arrays.

Instead of creating a new JSP we will go back and reexamine the JSP we created in 4.4.2, “Creating a JSP that uses JDBC to access Domino R5 data” on page 82 to see how data is passed.

Our JSP uses the `getParameter` method to keep the field contents after the JSP data is submitted to the server and to retrieve the data of the view the user wants to display. The response object is used to create the dynamic parts of the response page. This is shown in the following code extract.

```
<%
    try {
        Class.forName ("lotus.jdbc.domino.DominoDriver");
        Connection objCon = DriverManager.getConnection("jdbc:domino://" +
            request.getParameter ("db") + "/" +
            request.getParameter ("server"));
        java.io.PrintWriter pw = response.getWriter();
        Statement objGetNames = objCon.createStatement();
        ResultSet objResNames = objGetNames.executeQuery ("SELECT * FROM " +
            request.getParameter ("view"));
        // print lines
        String sLine = "";
        while (objResNames.next()) {
            sLine = "<TR>";
            for (int iCol = 1;
                iCol <= objResNames.getMetaData().getColumnCount(); iCol++) {
                Object objCell = objResNames.getObject(iCol);
                if (objResNames.isNull()) {
                    sLine = sLine + "<TD> </TD>";
                } else {
                    sLine = sLine + "<TD>" + objCell.toString() + "</TD>";
                }
            }
            sLine = sLine + "</TR>";
            // This command writes one line to the browser window:
            pw.println(sLine);
        }
        objCon.close();
    } catch (Exception e) {
        System.out.println (e.getMessage());
    }
}%>
```

In Figure 69 on page 118 you can see an example in Domino Designer of a form that will invoke our JSP from the previous chapter. Note that the field names are all lowercase to match the parameter names that the JSP requires.

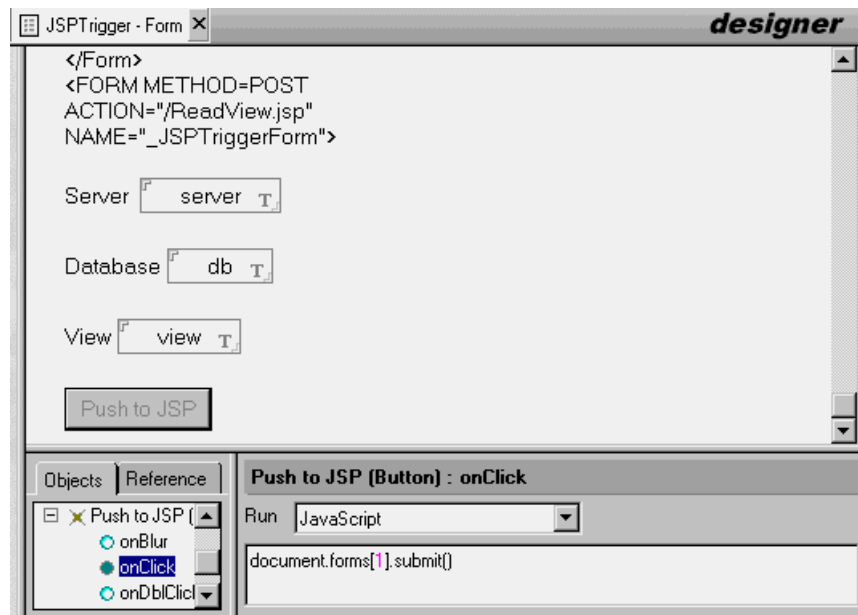


Figure 69. Domino form that calls a JSP

Note: In the real world you would probably never call a JSP from a Domino form to have data read from a Domino database. However, you may need to add Domino functionality to a Web application where a JSP is used to connect to some external data source. In that case it might make sense to collect the required parameters for the JSP in a Domino form and then pass them on to the existing JSP instead of rewriting the logic in the JSP.

5.3 Calling Enterprise Java Beans that are managed by WebSphere

In 3.3, “Enterprise JavaBeans” on page 52 we explained what Enterprise Java Beans are and how they are used. This chapter describes in detail how a Java program or agent (a client application) can invoke an EJB.

WebSphere EJBs can be used via their Remote Method Invocation (RMI) interface from Java programs, like Java applications, applets, servlets, and Domino R5 Java agents. In addition, they have a CORBA interface, which enables any program that can use CORBA and the Internet Inter-ORB Protocol (IIOP) to call WebSphere EJBs.

The Advanced Edition of WebSphere we used for our tests only supports the RMI interface. The Enterprise edition contains the IBM Component Broker which adds support for the CORBA/IIOP Interface.

5.3.1 Prerequisites for calling WebSphere EJBs using RMI

In every Java program that works as a client for WebSphere EJBs you use the Java RMI classes and some additional JDK and WebSphere classes to call the EJBs. The steps we describe in this section ensure that you can access them.

5.3.1.1 Include required Java archives into your classpath

An EJB client must have access to certain Java archives which are provided by WebSphere. Servlets that run in the WebSphere environment automatically have access to these archives via the WebSphere classpath.

Make sure that these three files can be accessed in your development environment and in your runtime environment:

- `ujc.jar`
- `ejs.jar`
- `iioptools.jar`

You find them in the `AppServer\lib` subdirectory of your WebSphere directory. They contain the Websphere EJB client environment. In our case we added them to the `NOTES.INI` file (`JavaUserClasses`). We already had them available in WebSphere, where they are added to the `admin.config` file (`com.ibm.ejs.sm.adminserver.classpath`) during installation. We also had them in VA Java because we added the IBM WebSphere Test Environment.

If your EJB client is not a WebSphere servlet, you need one additional archive per EJB you are calling. The WebSphere application server creates it when you deploy the EJB and places it in the subdirectory `deployedEJBs` of your WebSphere directory. WebSphere adds the prefix “Deployed” to the name of the original archive. That means after you deploy an EJB that is packaged in an archive named `Account.jar`, you will find a file named `DeployedAccount.jar` in the `deployedEJBs` directory.

In addition to all EJB classes you need at execution time, this file contains the class for the client stub. Therefore, you will not be able to look up the client stub as described in 5.3.2, “Getting the client stub via the naming service” on page 120 unless your EJB client has access to this file in the runtime environment. In the development environment you should use the original EJB classes instead of the classes in this file.

5.3.1.2 Importing required Java packages

Each WebSphere EJB client must import the following packages:

- java.util - This package contains some utility classes the WebSphere EJB client uses.
- java.rmi - This package contains classes for remote method invocation (RMI). You have to include this package because, typically, Java EJB clients use the RMI interface of EJBs.
- javax.rmi - This package contains the PortableRemoteObject class required to get a reference to an EJB object.
- javax.ejb - This package contains the classes and interfaces defined in the EJB specification.
- javax.naming - This package is used by the naming service to get a reference to the EJB objects.

In addition, you need to import the packages that contain the EJB classes your client is interacting with. If you are calling the account example that comes with WebSphere, the import section of your client class looks like:

```
...
// Classes necessary for EJB access
import java.util.*;
import java.rmi.*;
import javax.rmi.*;
import javax.ejb.*;
import javax.naming.*;
//EJB classes that are used in this program
import com.ibm.ejs.doc.account.*;
...
```

5.3.2 Getting the client stub via the naming service

Each EJB has a home class that works as a stub of the EJB in the client. WebSphere provides a name service that finds the bean and returns its home object. This service uses IIOP for the client-server communication.

This example code creates the initial context necessary to access the name service:

```
sProviderURL = "iiop://yourhostname:900";
sNameService = "com.ibm.ejs.ns.jndi.CNInitialContextFactory";
Hashtable htEnv = new Hashtable();
htEnv.put(javax.naming.Context.PROVIDER_URL, sProviderURL);
htEnv.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, sNameService);
InitialContext ctx = new InitialContext(htEnv);
```

The steps covered in the sample code are:

1. Create a new Java Hashtable.
2. Set the property `javax.naming.Context.PROVIDER_URL` in the hashtable. It contains the URL of the name service. The URL consists of the string `"iiop://"` to indicate an IIOP connection, the host name of your WebSphere server, and the port. The IIOP port the server is listening to usually is 900.
3. Set the property `javax.naming.Context.INITIAL_CONTEXT_FACTORY` in the hashtable to the name of the class that implements the naming service. This class depends on the WebSphere server you are using. For the advanced edition you use `com.ibm.ejs.ns.jndi.CNInitialContextFactory`. For the enterprise edition you need `com.ibm.ejb.cb.runtime.CBCtxFactory`.
4. Create a new object of the class `javax.naming.InitialContext` and pass the hashtable as a parameter to the constructor of this class.

After you have created the initial context, use it to look up the EJB. In the advanced edition you use the name that you specified in deployment descriptor of the EJB. You can find it in the WebSphere administrative console as shown in Figure 70.

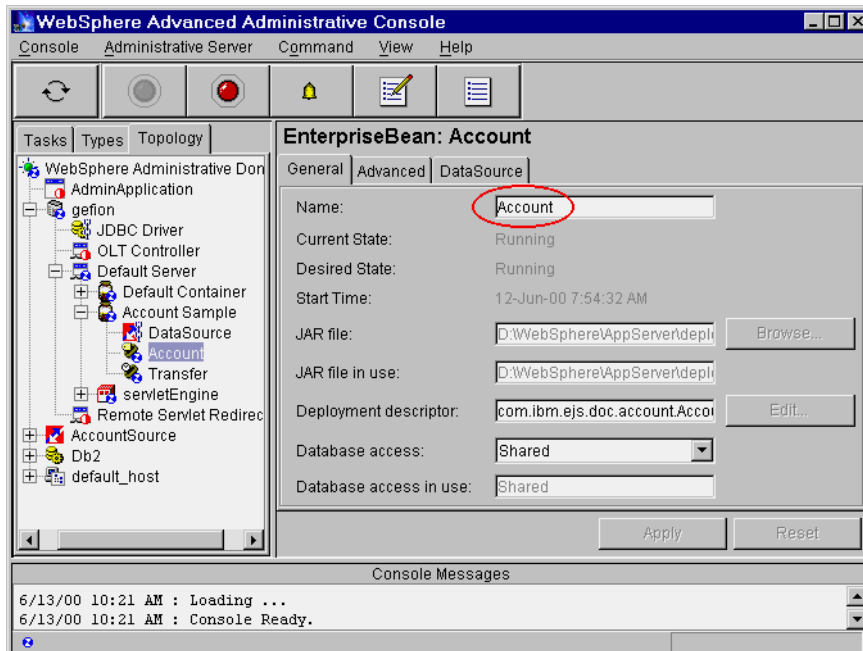


Figure 70. EJB name in the WebSphere administrative console

In the enterprise edition you add a prefix to the EJB name. The prefix depends on the EJB's setup and can be host/resources/factories/EJBHomes/, workgroup/resources/factories/EJBHomes/ or cell/resources/factories/EJBHomes/.

You should check the class of the object the name service returns using the narrow method of the javax.rmi.PortableRemoteObject class. This class throws a ClassCastException if the object that is returned by the name lookup is not an object of the intended class. A possible reason for this exception can be that the bean-specific archive is not accessible in the runtime environment. If so, you have to modify the setup as described in 5.3.1.1, "Include required Java archives into your classpath" on page 119.

The code for the EJB lookup and the class check looks like this:

```
Object objHome = ctx.lookup("Account");
ejbhAccount = (AccountHome) PortableRemoteObject.narrow
    ((org.omg.CORBA.Object) objHome, AccountHome.class);
```

5.3.3 Creating an EJB and calling the methods it provides.

After you have located the home object of an EJB, you can use it to create a new EJB or find an existing EJB. A create method is invoked to create an EJB object or a finder method is invoked to find an existing EJB object. The finder method is only valid for entity beans.

The code to create a bean looks like:

```
// Create the EJB.
Account ejbAccount = ejbhAccount.create(objKey, iTypeAcct, fBalance);
```

After the bean is created you can use all public methods it provides. For session beans you should handle the java.rmi.NoSuchObjectException. This exception is thrown when the session bean does not exist any longer. This can happen when the WebSphere server is stopped and restarted.

5.4 Using Enterprise Java Beans from a Domino R5 Java agent

In Domino Java agents an EJB can be invoked using several techniques. First we discuss in detail how to access an EJB directly from an agent. Since this requires the deployment of the Java class files used to all Domino servers the agent is running on, we also show how you can modify this agent to use an RMI server to avoid the requirement of "local deployment."

The third method of accessing an EJB from a Domino R5 agent is parsing the return page of a servlet using the Java URL classes. We explain the architecture of this method without a detailed code example.

If you are planning an agent that is called via a URL, you should consider the alternative of calling a servlet instead. As soon as your Web site is accessed by many users, the WebSphere servlet manager can handle requests much faster than the Domino R5 agent manager.

5.4.1 Invoking EJBs from Domino R5 Java agents directly

In this section we write a Domino R5 example background agent that calls the transfer EJB to transfer a fixed amount from one account to another regularly. The transfer EJB is part of the sample we described in 4.3, “Deploying the IBM WebSphere account example” on page 67. It models a funds transfer session that involves moving a specified amount between two instances of the Account EJB.

5.4.1.1 Creating a Domino R5 form and view

The example application uses Domino R5 documents to store the transfer orders. To be able to create these documents you create a Domino form (we called ours *TransferOrder*) that contains the following fields:

Table 5. Fields of the Domino form for transfer documents

Field name	Type	Formula
Amount	Number, Editable	
FromAccount	Text, Editable	
CurrentBalance1	Number, Computed when composed	0
ToAccount	Text, Editable	
CurrentBalance2	Number, Computed when composed	0
Message	Text, Computed when composed	" "

Create a view named Transfers that displays all transfer documents. Then create at least one transfer document.

5.4.1.2 Architecture of the Domino R5 Java agent

In a Domino R5 Java agent, you can access an EJB as described in 5.3, “Calling Enterprise Java Beans that are managed by WebSphere” on

page 118. However, you should create a separate Java thread that accesses the EJB as shown in Figure 71.

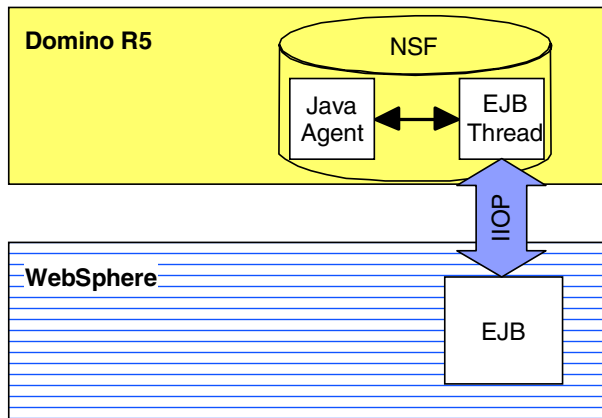


Figure 71. Architecture when invoking an EJB via a separate agent thread

If you try to connect to an EJB in the agent thread or a thread of the same group as the agent thread, the agent manager is not able to recycle the thread after the agent has stopped and displays this message on the Domino server console:

Addin: Agent error message: Error cleaning up agent threads

The Domino R5 server cannot free and reuse the memory the agent thread uses.

5.4.1.3 Processing Domino R5 documents using a Java agent

In VA Java select the **com.ibm.ejs.doc.sg245955** package in the **IBM Account example** project and create a new class. Enter **TransferAgent** as class name and **lotus.domino.AgentBase** as super class. Above the class definition add the statement:

```
import lotus.domino.*;
```

Create a new method named **NotesMain** that contains the following code to loop through the transfer definition documents:

```
public void NotesMain() {
    try {
        Session sesCurrent = getSession();
        Database ndbCurrent =
            sesCurrent.getAgentContext().getCurrentDatabase();
        View vwTransfers = ndbCurrent.getView("Transfers");
        Document docTransfer = vwTransfers.getFirstDocument();
```



```

        while (docTransfer != null) {
            docTransfer.replaceItemValue("Message", "processed");
            docTransfer.save(true,true,true);
            docTransfer = vwTransfers.getNextDocument (docTransfer);
        }
    } catch (Exception e) {
        System.out.println("Error in Transfer agent: " + e.getMessage ());
    }
}

```

So far, the only thing this agent does is read through all documents in the *Transfer* view and writes “processed” in the message field. We will come back and add more code later, but first we will create the class responsible for connecting to the Transfer EJB.

5.4.1.4 Creating the class for the EJB access thread

To create a separate thread, the agent must consist of at least two classes. The class for the separate thread that accesses the EJB extends `java.lang.Thread` and not `lotus.domino.NotesThread` because it does not contain any Domino objects.

In VA Java highlight the **com.ibm.ejs.doc.sg245955** package and create a new class called **TransferAgentEjbAccess**. Select **java.lang.Thread** as the super class. VA Java creates the class and its constructors. Since this class performs the EJB access, you import all packages that are mentioned in 5.3.1.2, “Importing required Java packages” on page 120. This is shown in the following code.

```

// Classes necessary for EJB access
import java.util.*;
import java.rmi.*;
import javax.rmi.*;
import javax.ejb.*;
import javax.naming.*;
//EJB classes that are used in this program
import com.ibm.ejs.doc.transfer.*;

```

Create the following variables at class level:

```

protected boolean bInit = false;
protected static TransferAgentEjbAccess objEjbAccess = null;
protected String sMessage = "";
private ResourceBundle resString =
ResourceBundle.getBundle("com.ibm.ejs.doc.client.ClientResourceBundle");
private TransferHome ejbhTransfer = null;
private InitialContext ctxCurrent = null;

```

```
private float fBalance1 = 0;
private float fBalance2 = 0;
```

Your VA Java workspace should now look similar to Figure 72.

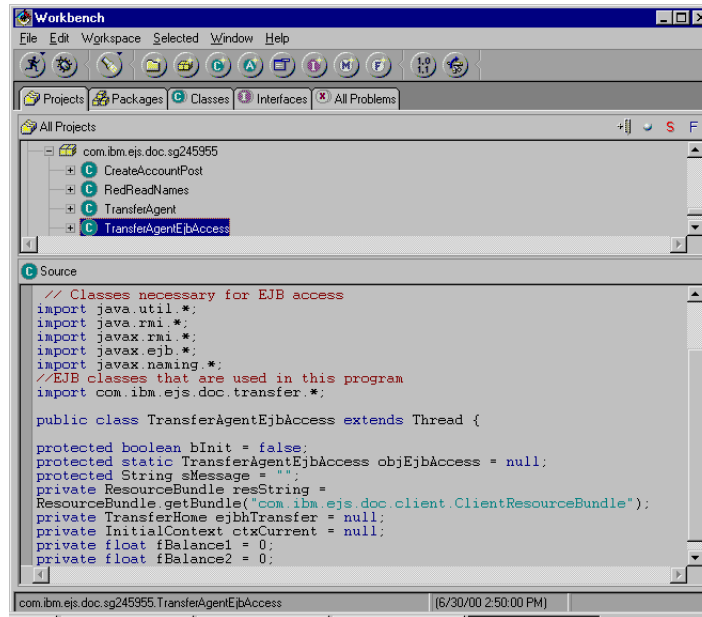


Figure 72. Class definition for *TransferAgentEjbAccess* in VA Java

To initialize the thread, use a method that creates an instance of the thread object and returns it. Create a method called *getInstance* and enter the following code:

```
public TransferAgentEjbAccess getInstance() {
    if (objEjbAccess == null) {
        objEjbAccess = new TransferAgentEjbAccess();
    }
    return objEjbAccess;
}
```

A thread class usually has a method called *run* that is running while the thread exists. In our example this method creates an instance of itself and the initial context. Then it starts a loop to keep the thread alive. Create the *run* method and enter the following code:

```
public void run() {
    getInstance();
    try {
        // Create the initial context.
    }
}
```

```

String sNameService = resString.getString("nameService");
Hashtable htEnv = new Hashtable();
htEnv.put(javax.naming.Context.PROVIDER_URL,
    "iiop://yourhostname:900");
htEnv.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
    sNameService);
ctxCurrent = new InitialContext(htEnv);
System.out.println ("Ejb proxy initialized");
bInit = true;
while (true) {
    sleep(3600000);
    System.out.println ("EJB access thread is still alive.");
}
} catch (Exception e) {
    sMessage = resString.getString("failuregenexp") + e.getMessage();
} finally {
    // free memory
    bInit = true;
    ctxCurrent = null;
    ejbhTransfer = null;
    Runtime.getRuntime().gc();
}
}

```

Note: The host name of your WebSphere server must be specified in the IIOP URL in the above method. An example of the URL is:

```
iiop://gefion.lotus.com:900
```

The other methods of the EJB access thread class usually provide the connection to the EJB and call the EJB methods. In this example you create a method that transfers a specified amount from one account to another. Add the following method to the TransferAgentEjbAccess class:

```

public boolean transferFunds(String sFromAccount,String sToAccount,
    float fAmount) {
    long lFromKey      = 0;
    long lToKey        = 0;
    if (ejbhTransfer == null) {
        try {
            // Get the context
            if (ctxCurrent == null) {
                sMessage = resString.getString("failuregenexp") +
                    "Context could not be created";
                return false;
            }
            // Get the home object
            Object objHome = ctxCurrent.lookup("Transfer");

```

```

        ejbhTransfer = (TransferHome)PortableRemoteObject.narrow(
            (org.omg.CORBA.Object)objHome,TransferHome.class);
    } catch (NamingException e) {
        sMessage = resString.getString("failureinit") + e.getMessage();
        e.printStackTrace();
        return false;
    } catch (Exception e) {
        sMessage = resString.getString("failuregenexp") + e.getMessage();
        e.printStackTrace();
        return false;
    }
}
if (ejbhTransfer == null) {
    return false;
} else {
    try {
        lFromKey = Long.parseLong(sFromAccount);
        lToKey = Long.parseLong(sToAccount);
        // create session bean
        Transfer ejbTransfer = ejbhTransfer.create();
        // invoke transfer method.
        ejbTransfer.transferFunds(lFromKey,lToKey,fAmount);
        fBalance1 = ejbTransfer.getBalance(lFromKey);
        fBalance2 = ejbTransfer.getBalance(lToKey);
        sMessage = resString.getString("successtransfer");
        return true;
    } catch (Exception e) {
        sMessage = resString.getString("failuregenexp") + e.getMessage();
        e.printStackTrace();
        return false;
    }
}
}
}

```

To get the balance from the private variables in the EJB access thread class, add this method to the TransferAgentEjbAccess class:

```

public Double getBalance(int iAccount) {
    if (iAccount == 1) {
        return new Double (new Float(fBalance1).doubleValue());
    } else {
        return new Double (new Float(fBalance2).doubleValue());
    }
}
}

```

Now the TransferAgentEjbAccess class contains all methods you need to access this class.

5.4.1.5 Adding more to the TransferAgent class

The **TransferAgent** class does not have a direct connection to any EJB. To create and use the EJB access thread you add the following three helper methods to the TransferAgent class:

```
private ThreadGroup getGroup(ThreadGroup thgrCurrent, String sGroupName) {
    // get a thread group with a specified name
    if (thgrCurrent == null) {
        return null;
    } else if (thgrCurrent.getName().equals(sGroupName)) {
        return thgrCurrent;
    } else {
        return getGroup(thgrCurrent.getParent(), sGroupName);
    }
}

private Thread getThread(ThreadGroup thgrCurrent, String sThreadName) {
    // get a thread with a specified name
    if (thgrCurrent == null) {
        return null;
    }
    Thread athActive[] = new Thread[thgrCurrent.activeCount()];
    for (int iThrCounter=0; iThrCounter<thgrCurrent.enumerate(athActive);
        iThrCounter++)
    {
        Thread thCurrent = athActive[iThrCounter];
        if (thCurrent.getName().equals(sThreadName)) {
            return thCurrent;
        }
    }
    return getThread(thgrCurrent.getParent(), sThreadName);
}

private TransferAgentEjbAccess getEjbAccessThread(ThreadGroup thgrAgent) {
    // get or create the EJB access thread
    try {
        // get the system thread group
        ThreadGroup thgrSystem = getGroup (thgrAgent, "system");
        if (thgrSystem == null) {
            return null;
        }
        // get EJB access thread
        TransferAgentEjbAccess objEjbAccess =
            (TransferAgentEjbAccess) getThread(thgrSystem, "ejbAcc");
        if (objEjbAccess == null) {
            // Thread not found => create new thread
            objEjbAccess = new TransferAgentEjbAccess(thgrSystem, "ejbAcc");
        }
    }
}
```

```

        objEjbAccess.start();
    }
    return objEjbAccess;
} catch (Exception e) {
    e.printStackTrace();
    return null;
}
}

```

Now you modify the NotesMain method of the TransferAgent class. It uses the getEjbAccessThread method to get a handle to the EJB access thread. Then it uses the EJB to transfer the specified amount from one account to another. After all Domino documents are processed it stops the EJB access thread.

```

public void NotesMain() {
    try {
        // Get the Domino context
        Session sesCurrent = getSession();
        Database ndbCurrent =
            sesCurrent.getAgentContext().getCurrentDatabase();
        View vwTransfers = ndbCurrent.getView("Transfers");
        // Create or get the EJB access thread
        TransferAgentEjbAccess objEjbAccess =
            getEjbAccessThread(Thread.currentThread().getThreadGroup());
        while (!objEjbAccess.bInit) {
            // Wait until the new thread is initialized
            System.out.println("Agent is waiting for EJB thread" +
                "initialization...");
            sleep(10000);
        }
        Document docTransfer = vwTransfers.getFirstDocument();
        while (docTransfer != null) {
            if (objEjbAccess.transferFunds(
                docTransfer.getItemValueString("FromAccount"),
                docTransfer.getItemValueString("ToAccount"),
                new Double(
                    docTransfer.getItemValueDouble("Amount").floatValue())) {
                docTransfer.replaceItemValue("CurrentBalance1",
                    objEjbAccess.getBalance(1));
                docTransfer.replaceItemValue("CurrentBalance2",
                    objEjbAccess.getBalance(2));
            }
            docTransfer.replaceItemValue("Message", objEjbAccess.sMessage);
            docTransfer.save(true, true, true);
            docTransfer = vwTransfers.getNextDocument(docTransfer);
        }
        objEjbAccess.stop();
    }
}

```

```

    } catch (Exception e) {
        System.out.println("Error in Transfer agent: " + e.getMessage ());
    }
}

```

5.4.1.6 Reusing the EJB access thread

If your agent is run often with short intervals in between, you can increase the performance dramatically by reusing the EJB connection. This is important especially in agents that are called via a URL or as WebQueryOpen or WebQuerySave agents of a form because the user will have to wait until the agent returns.

The EJB access thread keeps its connection to the EJB until it is stopped. If you do not stop the thread, it usually keeps alive until the HTTP task of the Domino R5 server is stopped.

In the agent you are creating in this chapter, the only code change that is necessary to achieve this is to delete the line:

```
objEjbAccess.stop();
```

from the NotesMain method of the TransferAgent class.

5.4.1.7 Deploy and test the Domino R5 agent

To deploy the agent code you have created you need to include the WebSphere archives in the classpath the agent manager uses. Unless you specify a classpath, Java agents can only use the standard JDK 1.1 classes, the Domino R5 classes and all classes that are part of the agent.

Although the Domino R5 server document has a classpath field in the Internet Protocols / Domino Web Engine tab, this field has no effect for Java agents. To set the classpath for Java agents, use the *JavaUserClasses* variable in the NOTES.INI file. Include a directory for Java class files and the standard WebSphere Java archives.

```
JavaUserClasses=d:\jdk117p\src;d:\websphere\appserver\lib\ujc.jar;d:\websphere\appserver\lib\ejb.jar;d:\websphere\appserver\lib\iiopools.jar
```

Note: In NOTES.INI the above JavaUserClasses values must all be on *one* line.

You are only able to access a thread group higher than the thread group of the agent if the agent code is available on the server hard disk locally. If the agent code is contained in the agent, the agent manager throws a SecurityException if you try to access another thread group.

If you are using the thread group of the agent for the EJB access thread, the agent manager cannot clean up the agent threads and you have a memory leak.

This behavior of the Domino R5 agent makes local deployment (on the file system) of the agent code the only option if you want to access an EJB in your agent. We will now demonstrate how to do this.

Export the classes `TransferAgent` and `TransferAgentEjbAccess` to a directory you specified in the `JavaUserClasses` variable. We exported to `d:\jdk117p\src`. In addition you must export the `ClientResourceBundle` class from the `com.ibm.ejs.doc.client` package, because it is used in the agent classes.

You must also add the classes of the transfer EJB archive to the `CLASSPATH`. They are in the `AppServer\deployedEJBs` subdirectory of your WebSphere directory in the `DeployedTransfer.jar` file. You can do this by modifying the `JavaUserClasses` variable in `NOTES.INI`. If you do not want to modify this variable for every EJB you access from Domino, you can extract the archive to a directory in the classpath. You can do this using a tool like WinZip. We did this and extracted everything from `DeployedTransfer.jar` to `d:\jdk117p\src`. Keep the relative path if you extract. For example, in our case the `Transfer.class` must be in the following path after extraction:

```
D:\jdk1.1.7\src\com\ibm\ejb\doc\transfer
```

Then you create a new agent in your Domino database to contain the Java `TransferAgent` code.

- Give the agent a name and select for it being a shared agent. We called our agent **TransferAgent**.
- In the Run options list box select **Imported Java**; click the **Import Class Files** button.

A new dialog where you can select the files to import opens.

- As **Base directory** specify:

```
d:\jdk117p\src
```

In the left pane you will then see the folder in the base directory.

- Expand **com -> ibm -> doc -> sg245955**

You should now see a list of Java classes under `sg245955`.

- Select **TransferAgent.class** and click **Add/Replace files**.

- This copies TransferAgent.class to the pane for current agent files. Since all other classes are accessed from the server's hard disk, you do not need to include them into the agent.

Click OK to return to the agent settings.

- Select hourly execution for testing purposes and keep the default value for the agent to act on *All new and modified documents since last run*.

Your agent definition should now look similar to Figure 73.

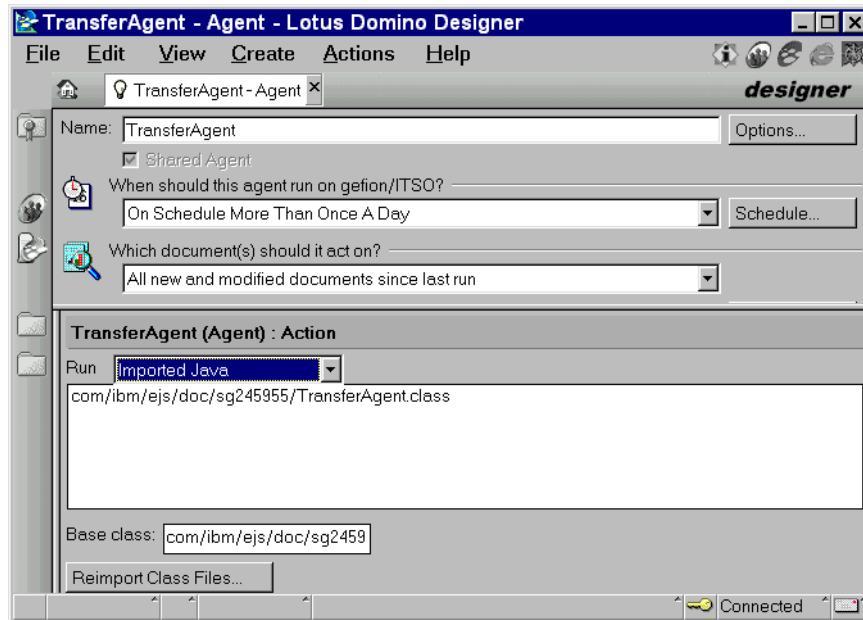


Figure 73. Settings for the TransferFunds agent

Save the agent and make sure that the Notes ID you are using has proper agent execution rights on your Domino server.

Now the agent is ready to perform all transfers specified in documents with valid data in the Domino database hourly. The output from one of our test runs looked like Figure 74 on page 134.

Amount	FromAccount	Balance1	ToAccount	Balance2	Message
3000	123	0	321	0	Exception: Account 123 does not exist.
333	456	0	654	0	Exception: Account 456 does not exist.
300	111111	4700	222222	1299	Transfer successful
999	222222	300	333333	1776	Transfer successful
30000	333333	0	111111	0	Exception: Insufficient fund in 333333

Figure 74. Transfer view after TransferAgent is run

You can see that we got both successful transfers as well as errors concerning non-existing account and insufficient funds to perform the transfer. You have to make sure that you have created the account numbers you specify. You can set up accounts using the example we installed in 4.3, “Deploying the IBM WebSphere account example” on page 67 using this URL:

<http://yourhostname/Account/Create.html>

A note about security

The thread we used for EJB access comes from the system thread group. Theoretically this might be considered a security exposure that would allow denial of service attacks because the thread continues to live after the agent has exited. The ability to create system threads from a Domino agent may be removed in a future release of Domino to avoid this exposure. As we have seen, however, for this techniques to work we must be able to add *JavaUserClasses* entries to the NOTES.INI file on the Domino server where the agent will run. We also need access to the file system to deploy our code locally (to avoid a security exception). Thus, we will not be able to deploy our agent code without already having full control of and access rights to the system anyway, so we do not consider this a dangerous technique.

We will now move on to the next example of accessing EJBs from Domino agents.

5.4.2 Using an RMI server to access EJBs

Instead of using a Java thread inside the Domino Java Virtual Machine (JVM) as we did in the previous example, we can use a separate Java program to communicate with an EJB. Since this program runs in a separate JVM, it can

only communicate with a Domino R5 agent via Remote Method Invocation (RMI). Therefore it must be written as an RMI server. The architecture of this solution is shown in Figure 75.

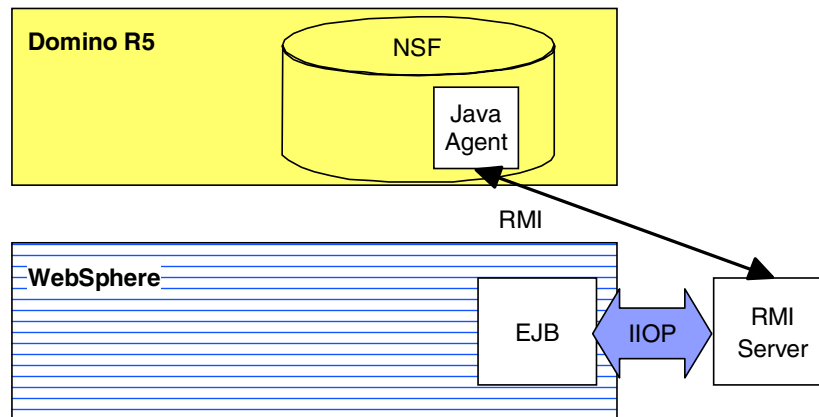


Figure 75. Architecture when accessing an EJB via an RMI server

Select this architecture if you want to separate the EJB access from your Domino R5 server or if you have different Java clients that use the same access methods for EJBs. In addition, this architecture prevents you from having to deploy your Java class files to all servers the agent runs on.

To create an RMI example, copy the agent classes you created in 5.4.1, “Invoking EJBs from Domino R5 Java agents directly” on page 123. Highlight the **TransferAgent** class and select **Reorganize -> Copy**. Keep the package name and specify **TransferAgentRMI** as the new name. Then copy the **TransferAgentEjbAccess** class to **TransferAgentRMIServer**.

5.4.2.1 Creating an interface for the server class

To create an RMI server, you have to define an interface containing all methods you want to call remotely. Each method in this interface must throw a `RemoteException`. Name your interface `TransferAgentRMIInterface`. Since you call the `transferFunds` method and the `getBalance` method of the remote server, the new interface contains these two methods. In addition you need a `getMessage` method to be able to retrieve the EJB messages.

```
import java.rmi.*;

public interface TransferAgentRMIInterface extends java.rmi.Remote {
    boolean transferFunds(String sFromAccount, String sToAccount, float
fAmount)
        throws RemoteException;
    Double getBalance(int iAccount) throws RemoteException;
}
```

```
String getMessage() throws RemoteException;
}
```

5.4.2.2 Implementing the RMI server class

The RMI server class must implement the interface you created and extend the `java.rmi.server.UnicastRemoteObject`. Modify the definition of the `TransferAgentRMIServer` class as follows:

```
public class TransferAgentRMIServer extends
java.rmi.server.UnicastRemoteObject
    implements TransferAgentRMIInterface {
    ...
}
```

Since the RMI server needs a different constructor method than a thread, you must delete all constructor methods, that means all methods that have names starting with `TransferAgentRMIServer`. The `getInstance` method can be deleted too.

The new constructor method of the RMI server establishes the connection to the EJB. This was the task of the `run` method in the thread. Rename the `run` method to `TransferAgentRMIServer` and change the code as follows:

```
public TransferAgentRMIServer(String sServer) throws RemoteException {
    super ();
    try {
        // Create the initial context.
        String sNameService = resString.getString("nameService");
        Hashtable htEnv = new Hashtable();
        htEnv.put(javax.naming.Context.PROVIDER_URL,
            "iiop://" + sServer + ":900");
        htEnv.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
            sNameService);
        ctxCurrent = new InitialContext(htEnv);
        System.out.println ("Ejb access initialized");
    } catch (Exception e) {
        sMessage = resString.getString("failuregenexp") + e.getMessage();
    }
}
```

Then add the `RemoteException` to the `transferFunds` and to the `getBalance` method and create the `getMessage` method with the following code:

```
public String getMessage() throws RemoteException {
    return sMessage;
}
```

All methods that are part of the interface must be implemented as defined in the interface.

Since the RMI server is a Java application, it has a main method. You create it with the code to start the RMI server:

```
public static void main(String args[]) {
    try {
        // connect to RMI registry
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        // create and bind RMI->EJB proxy object
        TransferAgentRMIServer objServer =
            new TransferAgentRMIServer(args[0]);
        Naming.rebind("//" + args[0] + "/TransferAgentRmiServer", objServer);
        System.out.println("Transfer agent RMI server bound");
    } catch (Exception e) {
        System.out.println("bind RMI server failed!");
        e.printStackTrace();
    }
}
```

In this example we assume that your WebSphere server and your RMI server run on the same host. If this is not the case, use different parameters for the WebSphere server that is passed to the class constructor and for the RMI server that is passed to the RMI naming class.

5.4.2.3 Generating the RMI stub and skeleton

To create the RMI communication two additional classes are necessary. Request VA Java to generate them automatically like this:

Highlight the **TransferAgentRMIServer** class and select **Selected -> Tools -> Generate RMI -> RMI Stub and Skeleton**.

This makes VA Java create the following two classes:

- TransferAgentRMIServer_Skel
- TransferAgentRMIServer_Stub

5.4.2.4 Modifying the agent class

Now you modify the TransferAgentRMI class to call the RMI server instead of creating an EJB access thread.

- Delete the three methods:
 - getGroup

- getThread
- getEjbAccessThread

These methods were used to access the EJB thread and we do not need them in this example.

- Then you modify the NotesMain method as follows:

```
public void NotesMain() {
    try {
        // Get the Domino context
        Session sesCurrent = getSession();
        Database ndbCurrent =
            sesCurrent.getAgentContext().getCurrentDatabase();
        View vwTransfers = ndbCurrent.getView("Transfers");
        TransferAgentRMIInterface rmiTransfer =
            (TransferAgentRMIInterface) java.rmi.Naming.lookup(
                "//yourhostname/TransferAgentRmiServer");
        Document docTransfer = vwTransfers.getFirstDocument();
        while (docTransfer != null) {
            if (rmiTransfer.transferFunds(
                docTransfer.getItemValueString("FromAccount"),
                docTransfer.getItemValueString("ToAccount"),
                new Double(docTransfer.getItemValueDouble(
                    "Amount")).floatValue())) {
                docTransfer.replaceItemValue("CurrentBalance1",
                    rmiTransfer.getBalance(1));
                docTransfer.replaceItemValue("CurrentBalance2",
                    rmiTransfer.getBalance(2));
            }

            docTransfer.replaceItemValue("Message", rmiTransfer.getMessage());
            docTransfer.save(true, true, true);
            docTransfer = vwTransfers.getNextDocument (docTransfer);
        }
    } catch (Exception e) {
        System.out.println("Error in Transfer agent: " + e.getMessage ());
    }
}
```

Again, remember to replace yourhostname in the code above with your server's host name. For example, we used *gefion.lotus.com*.

We are now ready to try out our code.

5.4.2.5 Deploying the RMI server and the agent

We will export our code and register our RMI server with the RMI registry.

- Export the server class, the skeleton class, the stub class and the interface to the Java directory of your RMI server.

In our case we are still using:

```
d:\jdk117p\src
```

Make sure that this directory and the WebSphere archives for the EJB client access are in the system classpath.

- Then you start the RMI registry, which is part of the Sun JDK, and your RMI server, by typing the following on the command line:

```
start rmiregistry
```

This may either simply start the registry as a background process or open another command prompt to run the process in. You will not get any feedback about it having started successfully; you just have to trust that this is the case. If a window was opened minimize it; do not close it!

- Now go back to the command prompt where you typed in the command and type the following substituting *yourhostname* with your own host name like *gefion.lotus.com*:

```
java com.ibm.ejs.doc.sg245955.TransferAgentRMIServer yourhostname
```

You should see the following message in the command prompt window:

```
Ejb access initialized
```

If you see this message the RMI server is running and can be reached via the RMI registry.

Now you must create a new agent called TransferAgentRMI with the same settings as the agent you created in 5.4.1.7, "Deploy and test the Domino R5 agent" on page 131.

- Click **Import Class Files...** in the agent definition and select the same Base directory again:

```
d:\jdk117p\src
```

- Expand the folders **com -> ibm -> ejb -> doc -> sg245955** and select the following classes:
 - **TransferAgentRMI**
 - **TransferAgentRMIInterface**
 - **TransferAgentRMIServer_Stub.**

- Click **Add/Replace files** and then **OK** to return to the agent definition.

The base class of the agent is TransferAgentRMI. This should be specified automatically in Domino Designer.

- Specify an hourly schedule for the agent and save it.

Your agent should now look similar to Figure 76.

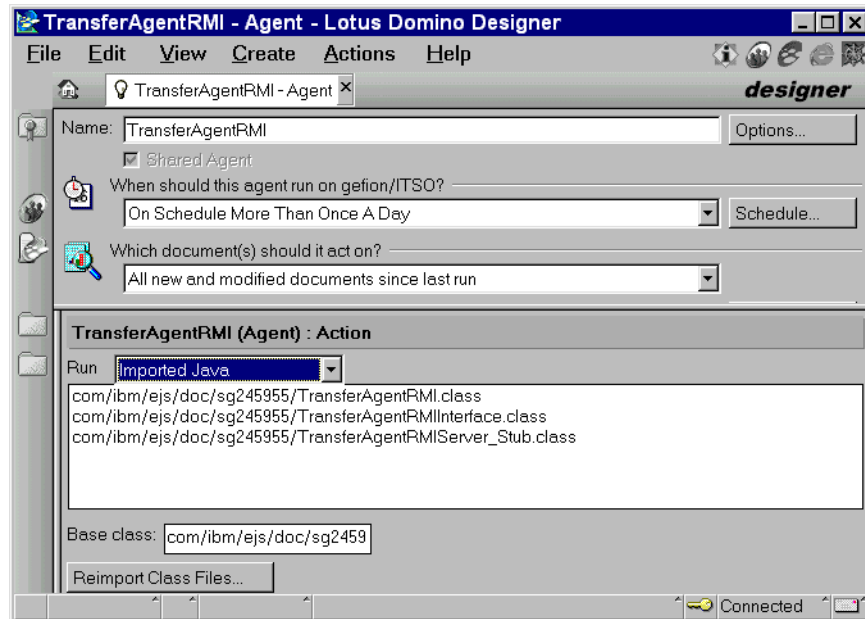


Figure 76. Settings for the TransferFunds agent with RMI access

This agent can run on any Domino R5 server that has access to the RMI server via the network. Enable the agent and wait for it to run.

When the agent has run, the output in your Domino database will be similar to what we had with our locally deployed agent as shown in Figure 74 on page 134. However, if your transaction triggers any error messages you will be able to see those in the command prompt window where our RMI server is running. In our case we got the output from the RMI server shown in Figure 77 on page 141.


```
Command Prompt - java com.ibm.ejs.doc.sg245955.TransferAgentRMIServer tyr.lotus.com
D:\>start rmiregistry
D:\>java com.ibm.ejs.doc.sg245955.TransferAgentRMIServer tyr.lotus.com
Ejb access initialized
Transfer agent RMI server bound
javax.ejb.FinderException: Account 123 does not exist.
javax.ejb.FinderException: Account 456 does not exist.
com.ibm.ejs.doc.account.InsufficientFundsException: Insufficient fund in 333333
```

Figure 77. Output from TransferAgentRMIServer

We have now tried two ways to access an EJB from a Domino agent. You can also do it with a servlet as intermediary which we discuss in the next section.

5.4.3 Invoking an EJB from a Domino R5 Java agent via a servlet

Using the Java URL Classes you can use a servlet from a Domino R5 agent by calling its URL, as shown in Figure 78. This architecture has the advantage that you only use the HTTP protocol. This prevent problems you could get with proxy servers or fire walls. In addition, you need not provide access for the WebSphere archives since EJB access is performed inside of WebSphere.

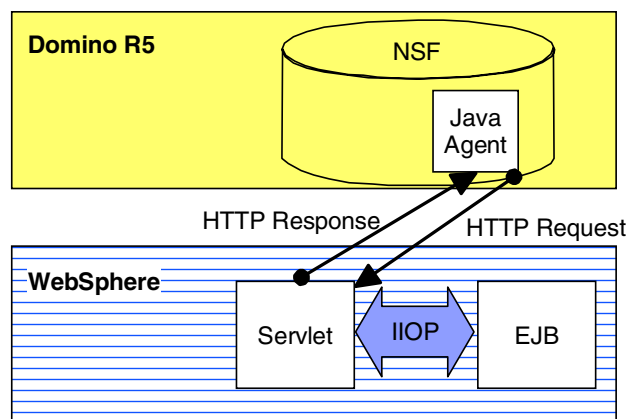


Figure 78. Architecture when accessing an EJB via a servlet

If you want to pass data to the servlet, you must use the parameters of the servlet URL. The parameters can be accessed within the servlet as described in 5.1.2, "Passing data to servlets in the URL" on page 108. To pass data from the servlet or from the EJB via the servlet to the Domino R5 agent, you can parse the page the servlet returns as shown a little further on.

If you want to access a servlet only via the Java URL classes, it should return a simple HTML page containing at least one keyword the Java code in the agent can search for.

The code below shows a sample Domino R5 agent that calls a servlet and searches for the string "Message" in the result returned. The agent then display the rest of the page the servlet is delivering until it reaches the </body> HTML tag. This is achieved by the following code:

```
public void NotesMain() {
    // get writer to browser
    PrintWriter pw = getAgentOutput();
    String sMessage = "";
    try {
        // get domino agent context
        Session sesCurrent = getSession();
        Document docCon = sesCurrent.getAgentContext().getDocumentContext();
        // create URL and open connection
        URL url = new URL("http",
            docCon.getItemValueString("Server_Name"),
            "/servlet/yourervletclass?yourparameters");
        URLConnection urlCon = url.openConnection();
        urlCon.connect();
        // parse page that is returned by the servlet
        boolean bNextToken = false;
        InputStream inp = urlCon.getInputStream();
        Reader rea = new BufferedReader(new InputStreamReader(inp));
        StreamTokenizer tokenizer = new StreamTokenizer(rea);
        while (tokenizer.TT_EOF != tokenizer.ttype) {
            tokenizer.nextToken();
            if (tokenizer.TT_WORD == tokenizer.ttype) {
                if (bNextToken) {
                    // add string to return string
                    if (sMessage == "")
                        sMessage = tokenizer.sval;
                    else
                        sMessage = sMessage + " " + tokenizer.sval;
                }
                else if (tokenizer.sval.equals("Message")) {
                    // message string found => read following text
                    bNextToken = true;
                }
            }
        }
        pw.println(sMessage);
    }
    catch (Exception e) {
```

```

        pw.println (sMessage + "[<BR>]Error: " + e.getMessage ());
    }
}

```

If you already have a servlet as an EJB client and it returns an HTML page that is relatively easy to parse, this is a simple way to connect to an EJB from a Domino agent.

5.4.3.1 Getting to an EJB from Domino - pros and cons

We have now discussed three different way to access EJBs from Domino agents, and you may wonder which method to pick. At the time of this writing, getting from Domino to EJBs is such a new functionality that there aren't too many best practices established yet, but we can summarize some of the facts about the different methods.

Direct access

Direct access from a Domino agent to an EJB currently requires a separate thread to do it, from a thread group other than the agent thread's. This in turn requires that Java classes used reside on the local file system, which then requires more administrative work. Future releases of Domino may also contain changes that makes this technique stop working, but for now it is the only way to access an EJB directly from a Domino agent.

Access via RMI

Using an RMI server to access the EJB on behalf of a Domino agent RMI client adds some flexibility to your solution in that the Domino agent can run on any Domino server that can access the WebSphere server over the network. However, the RMI server is single-threaded and can thus only process one request at a time, which may be ok for agents but isn't terribly scalable. Connecting using RMI across firewalls or proxies may require their reconfiguration and your security administrator may not allow that. You also have to consider the administrative overhead of having the RMI registry running on a machine in your network.

Access via servlet

Putting a servlet in between your Domino agent and the EJB it wants to work with basically adds more development work and all data has to be exchanged as text. However, the servlet can be managed using WebSphere's administrative console and it is accessed using HTTP, which normally does not require any reconfiguration of proxies and firewalls, so the administrative cost will not be as great as with the other methods.

5.4.4 Summary

In this chapter we have covered the different ways you can use WebSphere from Domino R5. We have discussed:

- Invoking servlets from Domino

We have passed data to servlets via the URL and by posting a Domino form with field values.

- Calling JavaServer Pages from Domino

This included differences between servlets and JSPs in passing data to them, and we have showed a simple example of invoking a JSP with data from a Domino form.

- Calling Enterprise Java Beans from Domino

We discussed access to EJBs from Domino agents through direct access, through remote method invocation and via a servlet.

Chapter 6. Authentication and authorization

For certain Web applications, security is of paramount concern. Both WebSphere and Domino provide strong support for securing application access and data. Both products implement security mechanisms which involve verifying user identity (authentication) and allowing access to protected resources only to designated users (authorization). However, the WebSphere and Domino security architectures are different. For example, WebSphere's access control model is based on capabilities associated with a user; these define what a user can do (read, write, execute, and so on) with specified objects. Domino's access control model is resource-based, limiting actions users can perform according to entries in the Access Control List (ACL) stored with the database. It is further refined by document- and field-level access controls which can refine the basic ACL controls. Providing security for an application supported across both products requires establishing user identity and access controls in both products.

One aspect of the Domino and WebSphere security models, the directory (referred to as the "user registry" in WebSphere), can be made common. This is possible since both Domino and WebSphere support the Lightweight Directory Access Protocol (LDAP) for directory access. The Domino directory component provides for LDAP access and WebSphere can be set up to use an LDAP-accessible directory as its user registry.

In the following sections, we describe our experiments with Domino and WebSphere security support using a shared directory:

- Setting up a shared LDAP directory for Domino and WebSphere
 - Using Domino Directory
 - Using IBM SecureWay directory
- Using basic authentication in Domino and WebSphere
- Protecting a Web application
 - Domino Access Control Lists
 - Configuring an Enterprise Application in WebSphere

We assume the reader is familiar with the various features available for securing Web applications, such as HTTP Basic Authentication, Session Authentication, Secure Sockets Layer (SSL) encryption and Public Key Certificates. We also assume familiarity with directory concepts, especially those related to X.500 naming and LDAP.

6.1 Approaches for directory sharing

We see two basic approaches to establishing a common directory shared by Domino and WebSphere for authentication and authorization. One is to use the Domino Directory as the common directory for the WebSphere/Domino server. The other is to use another LDAP-compatible directory service as the common directory. For this approach, the directory service choices are preferably those explicitly supported by WebSphere and Domino. We chose the IBM SecureWay Directory product because it is offered as part of other IBM e-commerce product sets, for example, IBM WebSphere Commerce Suite.

Note that no matter what directory is used, security function depends on populating the directory with a consistent and complete set of users and groups on which access controls (authorization) can be based. We do not discuss issues concerning populating the directory in this book.

6.1.1 Sharing the Domino Directory

Using the Domino Directory as the common WebSphere/Domino directory service is attractive for those applications or environments where Domino is already established as the application base. Here, WebSphere would be used to complement Domino, providing support for those parts of an overall Domino application which require handling large numbers of transactions or guaranteed data integrity.

Current Domino Directory support allows Web users (as opposed to those using Notes clients to access the Domino server) and groups to be defined in the directory, along with credentials necessary for authenticating Web access (such as the Internet password). The fact that the Domino Directory allows access via LDAP enables it to be easily established as the WebSphere user registry. Indeed, “Domino 5.0” and “Domino 4.6” are shown as choices for the Directory Type on the WebSphere administrative console panel where the user registry is specified.

6.1.2 Sharing another LDAP directory

As previously mentioned, the WebSphere product supports the use of any LDAP-accessible directory as its user registry. Domino also supports the use of a separate LDAP directory for Web user authentication and access control in conjunction with the Domino Directory. It is possible then to set up a separate LDAP directory server and configure both WebSphere and Domino to use it as the user registry. Doing so allows the users requiring access to a combined WebSphere/Domino server to be managed in a single directory.

Using a separate LDAP directory would be a good choice for application environments where WebSphere is used as the main application engine and Domino is used for a subset of functions. For example, a virtual banking application could be supported mostly by HTML, servlets, JSPs, or EJBs served by WebSphere, with Domino providing a loan approval workflow function. Using an LDAP directory is also appropriate where other products need to work with user authentication/authorization, such as IBM SecureWay Policy Director.

We investigated using the IBM SecureWay Directory product as a separate LDAP directory and configuring both WebSphere and Domino to use it as the user registry for user authentication and access control.

6.2 Using the Domino Directory

In this section we discuss setting up the Domino Directory as the user registry for Web user authentication and authorization on a combined WebSphere/Domino server. This involves:

- Having Domino set up to act as an LDAP server, and ensuring that a user in Domino Directory is set up as Administrator and has an internet password
- Enabling security in WebSphere and specifying Domino Directory as the directory to use

6.2.1 Configuring the Domino Directory for LDAP access

Domino R5 supports access to the Domino directory via LDAP. The only steps necessary to enable LDAP access to the Domino Directory are configuring the LDAP port in the Domino Server configuration document and starting the Domino LDAP server task.

It is also possible to extend the Domino LDAP schema if necessary. We did not need to do this to support our testing. If you need to do so, consult the *Domino R5 Administration Help* and *Administrator's Guide* for further direction.

You also have to make sure that there is a user defined in Domino Directory that has an internet password. In our case we created a Domino administrator ID named *WASAdmin* and assigned an internet password to it.

If you installed Domino as described in 2.6, "Installing Domino Server R5" on page 21, the Domino LDAP support is already set up and you can skip this

section and go directly to 6.2.2, “Configuring WebSphere to use the Domino Directory” on page 150.

To configure the LDAP port on a Domino server do the following:

1. Start Domino R5 Administrator client.
2. Open your server and select the **Configuration** tab.
 - a. Select **Server->Current Server Document** to display the server document.
 - b. Select the **Ports** tab.
 - c. Select the **Internet Ports** and **Directory** sub-tabs to display the LDAP port settings (see Figure 79).

The default LDAP standard and SSL port numbers (389 and 636) are shown. Ensure that at least the TCP/IP port status is set to *Enabled*.

(We recommend that for a production environment the TCP/IP port be disabled and the SSL port be enabled with at least “Name & password” authentication.) For our testing we used the settings shown in Figure 80 on page 149.

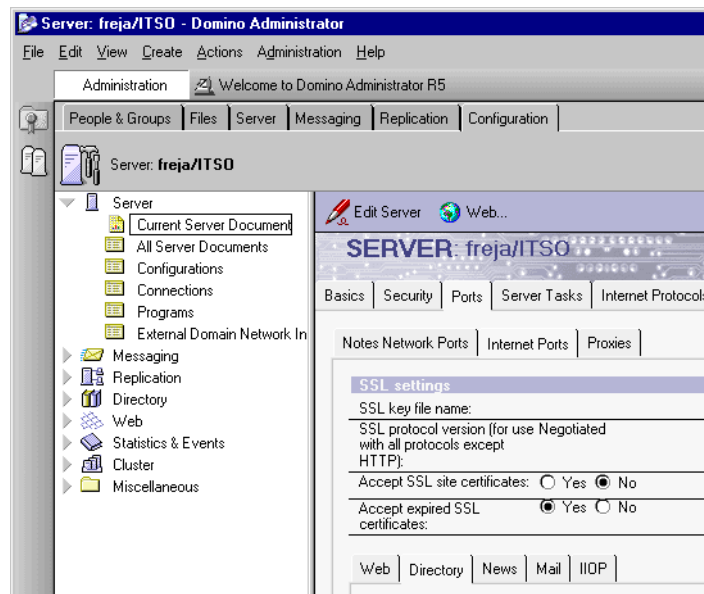


Figure 79. Domino Administrator client showing Internet Port settings

Web		Directory	News	Mail	IIOP
		Directory (LDAP)			
TCP/IP port number:		389			
TCP/IP port status:		Enabled			
Authentication options:					
Name & password:		Yes			
Anonymous:		Yes			
SSL port number:		636			
SSL port status:		Disabled			
Authentication options:					
Client certificate:		No			
Name & password:		No			
Anonymous:		Yes			

Figure 80. LDAP port settings for the Domino directory

Starting the Domino LDAP server task can be done from the Domino R5 Administrator client like this:

1. Open the Domino server, selecting the **Server** page.
2. Select the **Status** tab page.
 - a. Using the **Task->Start...** action, choose the LDAP Server entry and click **Start Task**.
 - b. Click **Done** to exit the tasks dialog.

The LDAP server task can also be started from the Domino console by entering the `load ldap` command. You may add the LDAP server task name to the `ServerTasks=` variable in the NOTES.INI file to start the task when the Domino server is started. If you picked LDAP support during the installation of Domino this entry already is on NOTES.INI.

If the LDAP server task is already running, you should restart it after making the changes to the server document. To restart the LDAP task from the Administrator client:

- Select the LDAP task from the **Server/Status** view.
- Right-click on the highlighted task, then select **Stop**.

Follow the preceding instructions to start the LDAP task. From the Domino console, first enter `tell ldap quit` followed by `load ldap`.

6.2.2 Configuring WebSphere to use the Domino Directory

Here we explain how to configure WebSphere to use the Domino directory via LDAP as its user registry.

For WebSphere to work with Domino Directory a security fix in addition to the WebSphere fixpack 1 is needed. The fix will be included in fixpack 2 for WebSphere. Look for it at this address:

<http://www.ibm.com/software/webservers/appserv/efix.html>

Make sure that the latest security fixes are installed before you proceed configuring WebSphere to use Domino Directory.

6.2.2.1 Disable security if it already is enabled in WebSphere

First, ensure that another directory is not already configured for WebSphere. (This is necessary since the configure task will attempt to bind to the newly specified LDAP directory to confirm the given Security Server ID name and password.) To check this, refer to Figure 81 on page 151 and do the following:

1. If it's not already running, start the WebSphere administrative console.
2. Select the **Tasks** tab in the left-hand window pane, then expand the **Security** branch of the tasks tree.
3. Select the **Specify Global Settings** task - you will see a help screen describing this task in the right-hand window pane.
4. Start the task wizard by clicking on the green button in the button bar.
5. The **General** tab page is shown; uncheck the **Enable Security** checkbox. (If it's already unchecked, skip the next step.)
6. Exit the WebSphere administrative console and restart the WebSphere administrative server (the IBM WS AdminServer service in the NT Services dialog.)

6.2.2.2 Specifying Domino Directory in WebSphere

Now we are ready to configure WebSphere to use the Domino Directory.

Ensure that the Domino server and the LDAP server task are started, then do the following:

1. Start (or restart) the WebSphere administrative console.
2. Start the **Specify Global Settings** task wizard.
3. Check **Enable Security** on the **General** tab page as shown in Figure 81 on page 151.

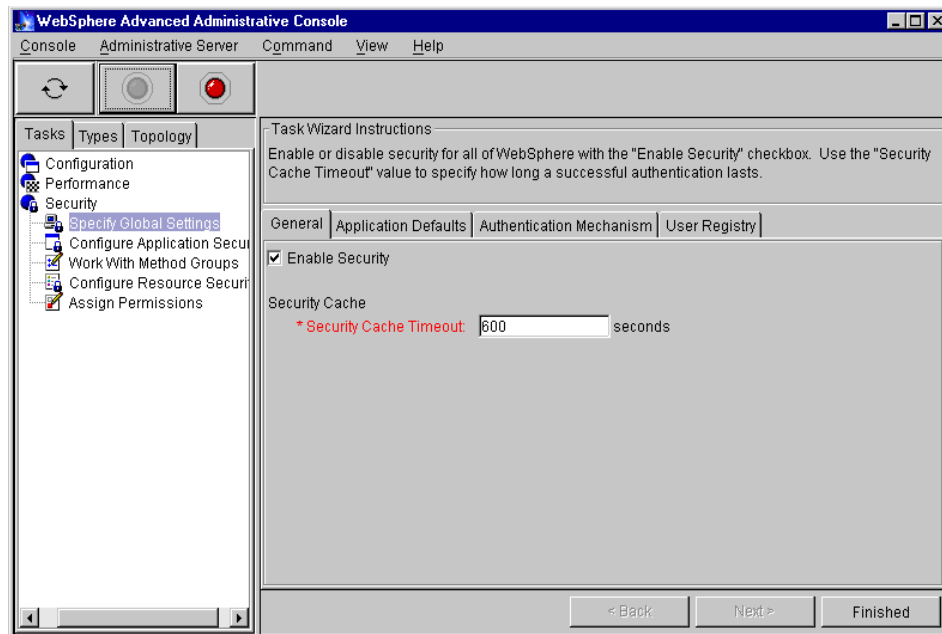


Figure 81. General settings for WebSphere security

4. On the **Application Defaults** tab page:

- Specify a Realm Name (we accepted our domain name *lotus.com* as the default suggestion)
- Choose **Basic (User ID and Password)** for Challenge Type (see 6.5.3, “Using basic authentication in our enterprise application” on page 176 for an explanation of these fields).

Note: As of this writing, basic authentication is the only challenge type supported by WebSphere Version 3.

5. On the **Authentication Mechanism** tab (Figure 82 on page 152) choose **Lightweight Third Party Authentication (LTPA)**.

Accept the default for the Token Expiration field.

If you have other IBM products that support LTPA you can import a set of encrypted keys to use for the authentication mechanism. This is not the case for us. In this case a set of encryption keys are generated and you will be prompted for a password to protect the keys. You will be asked for this password when you define which LDAP directory to use and you will also be asked for it if you regenerate the LTPA keys using the **Generate Keys** button.

You can leave the Enable Single Sign On (SSO) box unchecked.

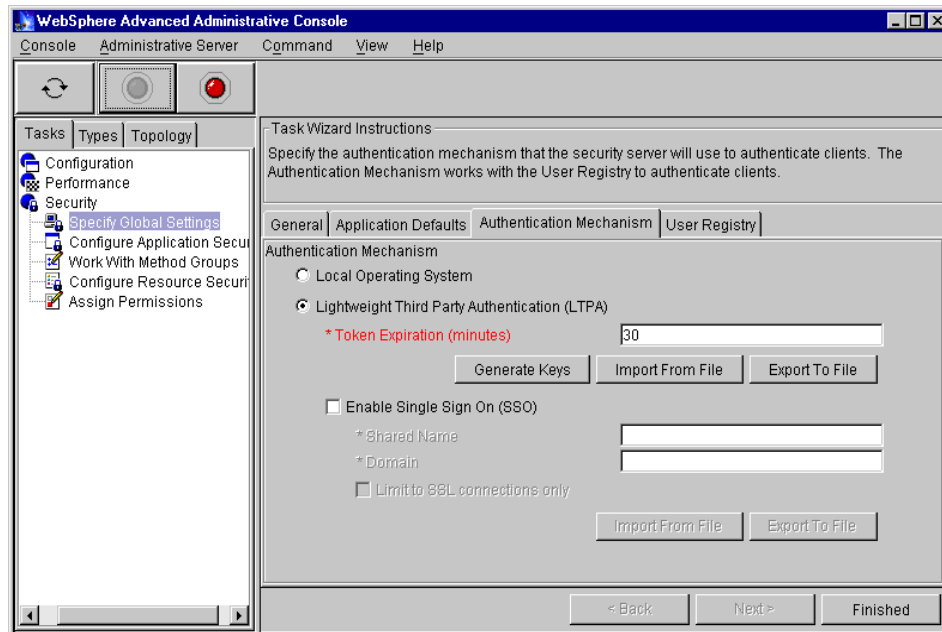


Figure 82. Authentication Mechanism settings

6. On the **User Registry** tab page:

- Set the Security Server ID to a user name defined in the Domino Directory with an Internet password. Specify it as an X.500 distinguished name (the format is a bit different from a Domino distinguished name) like this:

cn=John Doe,o=ITSO

- Set the Security Server Password to the Internet password of the user specified for Security Server ID.

The Security Server ID and Password are used to authenticate access to the WebSphere administrative console. We used the name of the Domino server administrator.

Note: WebSphere will attempt to do an LDAP bind to the name specified in Security Server ID when this task completes, so the name and password you choose must exist in the Domino Directory.

- Choose **Domino 5.0** as the directory type.
- Set the host name to the fully qualified host name of the Domino server, like:

tyr.lotus.com

- Leave all other fields blank. (You need to specify the Port field if you changed the default Domino LDAP port number.)

Figure 83 shows how our User Registry tab page looked after the steps we just went through. There is still one more step needed before we are done.

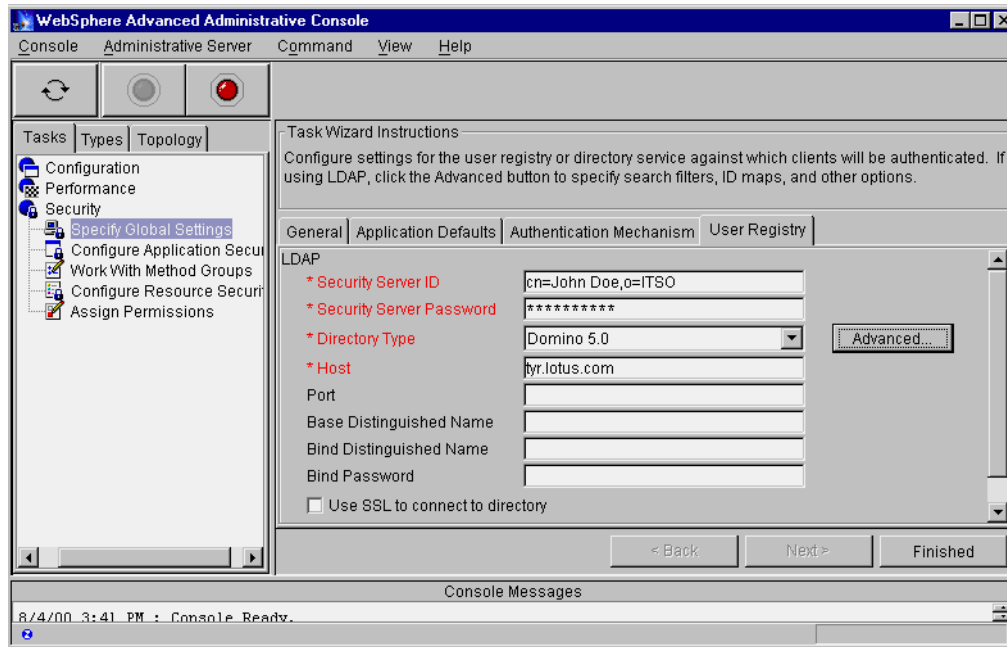


Figure 83. User Registry settings for the Domino directory

7. On the User Registry page click the **Advanced** button to view the defaults for how WebSphere will query the Domino directory. You must verify these settings against the schema used by your Domino Directory. In Domino R5.0.4 the object class `ePerson` is not part of the default LDAP schema for Domino Directory so we made the following changes:

- We changed the **User Filter** setting to specify `person` as the object class instead of `ePerson`:

```
(&(uid=%v)(objectclass=person))
```

The User Filter is used as the LDAP search filter when WebSphere searches for user names in the Assign Permissions task.

- We also changed the **User ID Map** setting to `*:cn` instead of `ePerson:shortname`

The User ID Map is the object class::attribute whose value is shown as the user name.

Figure 84 shows the changes we made.

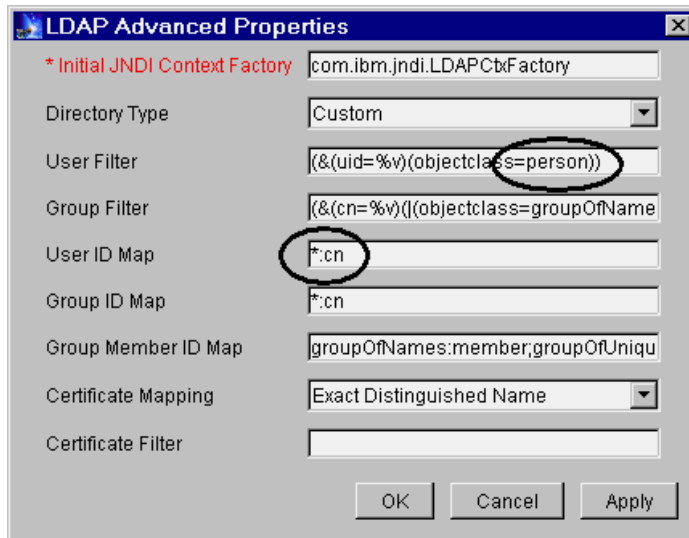


Figure 84. LDAP Advanced Properties for a Domino directory

Click **OK** to save your changes and return to the User Registry page.

Note: The Directory Type (which we specified as Domino 5.0) changed to Custom after we modified the Advanced settings as explained above.

8. Finally, click the **Finished** button. The configure task modifies the security configuration and reports its results. (If you are prompted for an LTPA password, specify one.)
 - If unsuccessful, you will see error messages in the console messages area. Check each of the fields discussed previously, especially the Security Server ID and Password fields, and ensure they are correctly set.
 - If successful, you will see a message box indicating that the administrative server must be restarted for the changes to take effect.
- Restarting the administrative server and stopping the console can be done by right-clicking on the host node in the **Topology** tree view and selecting **Restart**. You will need to start the administrative console again manually.
- Note:** Starting the WebSphere administrative service with security enabled will take longer than it did before you enabled security.

When the administrative server and console are restarted after enabling security with the Domino directory as the user registry, you are prompted to enter a name and password. Enter the name and password you specified for the Security Server ID and Password.

Having set up WebSphere to use Domino Directory, the next step in enabling authentication and authorization in your application is to make sure that basic authentication (which was the only challenge type that worked in the combined product set we worked with) is set up. This is described in 6.5.3, “Using basic authentication in our enterprise application” on page 176.

However, we will first describe how to set up WebSphere and Domino if you want to use another LDAP directory, in our example IBM SecureWay, to protect your Domino and WebSphere application.

6.3 Using another LDAP directory

In this section we discuss setting up a separate LDAP directory as the user registry for Web user authentication and authorization to a combined WebSphere/Domino server. In particular, we show how to configure the IBM SecureWay Directory product for this purpose and how to configure both WebSphere and Domino to use the SecureWay directory.

6.3.1 Installing and configuring the IBM SecureWay Directory

At the time of this writing, IBM SecureWay is available for downloading at no cost from IBM at this Web address:

<http://www-4.ibm.com/software/network/directory/downloads/>

We installed the IBM SecureWay Directory Version 3.1.1 product for Windows NT following the instructions given in the Installation and Configuration Guide (see <http://www.ibm.com/software/network/directory/library/>). We will not go into detail here since the manual explains the procedure. An overview of the installation steps, and our particular specifications, is as follows:

1. Set the directory administrator name and password.

We used `cn=db2admin` for the name and the same password associated with this NT user name. Note the use of the X.500 Distinguished Name (DN) format to specify the name.

2. Create the directory DB2 database.

We chose the **Create default database** option, which created a separate database manager instance and file tree in the NT file system.

3. Configure a Web server for directory administration.

We selected **Lotus Domino** as the Web server and specified the full path name to the Domino data directory for the httpd.cnf file. (SecureWay establishes its directory server administration client as a CGI program invoked from a supplied HTML page. The Pass statements necessary to point the Domino server to the HTML and CGI program are added to the httpd.cnf file.)

Initial configuration of the SecureWay LDAP directory is done via the administration Web interface (<http://<host>/ldap/index.html>). Log on using the DN of the directory administrator, in our case: `cn=db2admin`.

Any name suffixes for the directory tree must be created (usually these suffixes will be for the top-level organization and country entries). We chose to use the sample directory tree supplied with SecureWay, so we created an `o=ibm, c=us` suffix. Then we loaded the sample tree from the provided LDIF file (*sample.ldif*) using the *ldif2db* command line utility.

Next, we made sure that the Domino LDAP server was not running and modified the NOTES.INI file to remove LDAP from the Domino server task startup list.

Finally, we started the SecureWay LDAP directory server using the Web administration interface at the URL <http://<your host>/ldap>. For us it was:

<http://freja.lotus.com/ldap>

In the SecureWay Web administration interface you select **Server -> Startup/Shutdown** to start the server.

Note: You can also start and stop the directory server using the Services page in the Control Panel on the Windows NT machine where SecureWay is installed.

To check the directory server operation, we started the SecureWay Directory Management Tool (DMT) found in the IBM SecureWay Directory program group and added our server address. The DMT is basically an LDAP client with a GUI and functions for viewing and modifying the contents (schema and entries) of the directory. Note that in order to modify the directory contents you must bind to the directory as an administrative user (in our example, as `cn=db2admin`).

We used the Directory Management Tool to add passwords to the person we had imported into SecureWay from the sample file.

We are now ready to specify SecureWay as our LDAP directory in WebSphere.

6.3.2 Configuring WebSphere to use the SecureWay directory

Configuring WebSphere to use IBM SecureWay involves:

- Making sure security isn't already enabled using another directory
- Entering the SecureWay-specific information using the WebSphere administrative console

6.3.2.1 Disable security if it already is enabled in WebSphere

When specifying security for WebSphere we first have to make sure that previous security settings are disabled using the following steps:

1. Start the WebSphere Administrative Console.
2. On the **Task** tab start the **Security -> Specify Global Settings** task.
3. Uncheck the **Enable Security** checkbox (see Figure 81 on page 151).
4. Restart the WebSphere administrative server and console. (See "Configuring WebSphere to use the Domino Directory" on page 150 for an explanation of why this is needed.)

6.3.2.2 Specifying IBM SecureWay Directory in WebSphere

Before starting the **Specify Global Settings** task in the WebSphere administrative console, ensure that the SecureWay directory server is started. Then follow these steps:

1. Make sure the WebSphere administrative console is started.
2. On the **Task** tab start the **Specify Global Settings** task wizard.
3. Check **Enable Security** on the **General** tab page (Figure 81 on page 151).
4. On the **Application Defaults** tab page, specify a Realm Name. You may accept the default suggestion. For us the default value was *lotus.com*, which we used.
 - Choose **Basic (User ID and Password)** for challenge type.
(see 6.5.3, "Using basic authentication in our enterprise application" on page 176 for an explanation of these fields).

Note: As of this writing, basic authentication is the only challenge type supported by WebSphere Version 3.
5. On the **Authentication Mechanism** page (Figure 82 on page 152) choose **Lightweight Third Party Authentication (LTPA)**.
 - Accept the default for the Token Expiration field.
 - If you are prompted for an LTPA password for the set of encryption keys to be used by the LTPA mechanism, specify a password. You will be

asked for this password if you regenerate the LTPA keys using the **Generate Keys** button.

- You can leave the Enable Single Sign On (SSO) box unchecked.

6. On the **User Registry** tab page, set the **Security Server ID** to the DN of the SecureWay directory administrator. For us, it was:

`cn=db2admin`

Note: Using the DN is necessary, otherwise the configure task will fail.

- Set the **Security Server Password** to that of the directory administrator.

The Security Server ID and Password are used to authenticate access to the WebSphere Administrative Console. The name must exist in the SecureWay directory.

- Choose **SecureWay** as the directory type.
- Set the host name to the fully qualified host name of the directory server. In our case it was

`freja.lotus.com`

- Set the Base DN to the organization level of the directory tree. We specified `o=ibm,c=us` for our directory.

Figure 85 on page 159 shows our settings on the User Registry tab page.

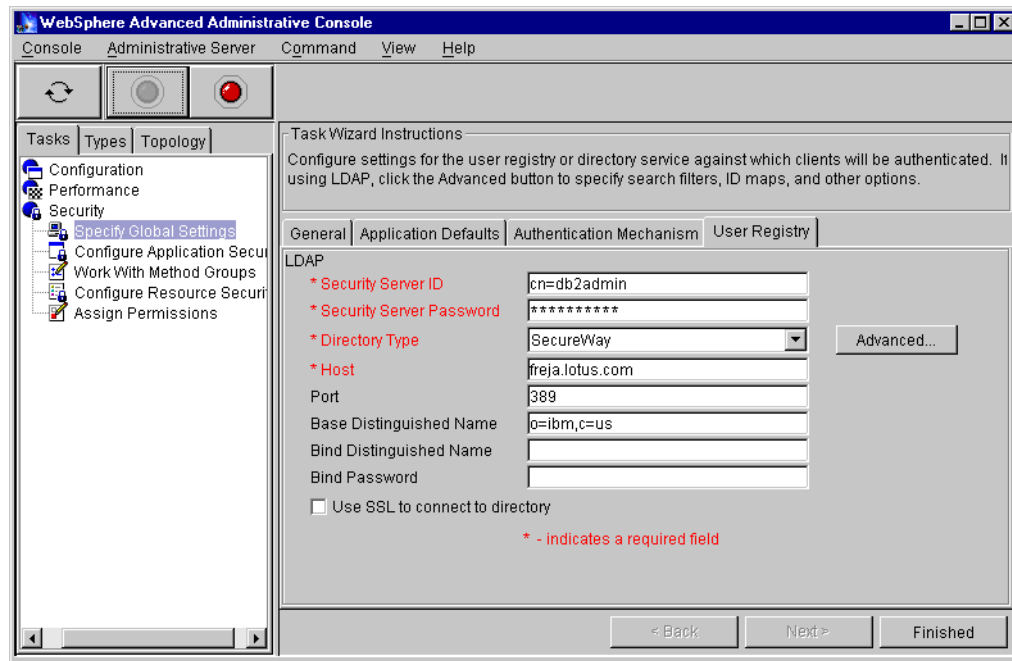


Figure 85. User Registry settings for SecureWay directory

7. Click the **Advanced** button to view the defaults for how WebSphere will query the SecureWay directory (Figure 86 on page 160).

These settings should be verified against the schema used by the SecureWay directory. Since we set up the directory from the sample data, we needed to change the User Filter field to specify last name (`sn`) as the search attribute and `person` as the object class. We also modified the User ID Map field to specify `*:cn;`; this controls how the user search results are displayed. See Figure 87 on page 160 for the modified search specifications.

Note: The Directory Type field changes to `Custom` because these settings were modified.

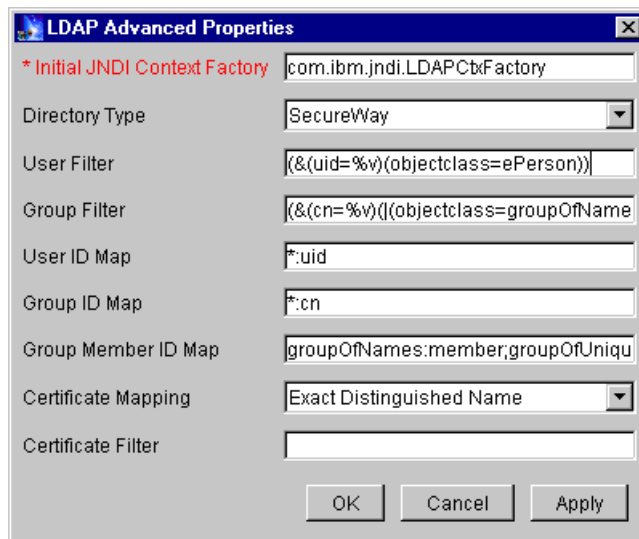


Figure 86. Default LDAP properties for SecureWay directory

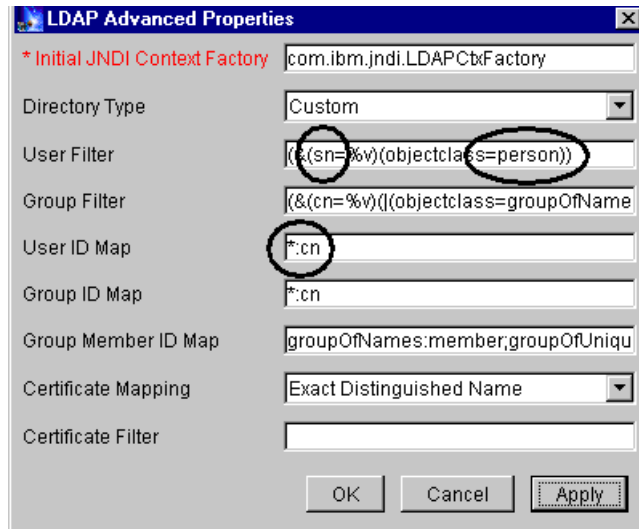


Figure 87. Modified LDAP properties for SecureWay directory

Click **OK** to save your modifications and return to the User Registry page.

8. Finally, click the **Finished** button.

The configure task modifies the security configuration and reports its results. If successful, you will see a message box indicating that the

administrative server must be restarted for the changes to take effect. If unsuccessful, you will see error messages in the console messages area. Check each of the fields discussed previously and ensure they are correctly set.

When the WebSphere administrative server and console are restarted after enabling security with the SecureWay directory as the user registry, you are prompted to enter a name and password. Enter the name and password you specified for the Security Server ID and Password. Remember you have to use the distinguished name form, like cn=db2admin.

6.3.2.3 Verifying correct setup

To verify that the SecureWay directory is being used by WebSphere, check the global security settings again.

- Start the WebSphere administrative console.
- Select the **Security->Assign Permissions** task, select one of the permissions and click **Add** to open the Search dialog, then verify that the user and group name search function finds the users and groups in the SecureWay directory. See Figure 88 for a sample of what this dialog shows.

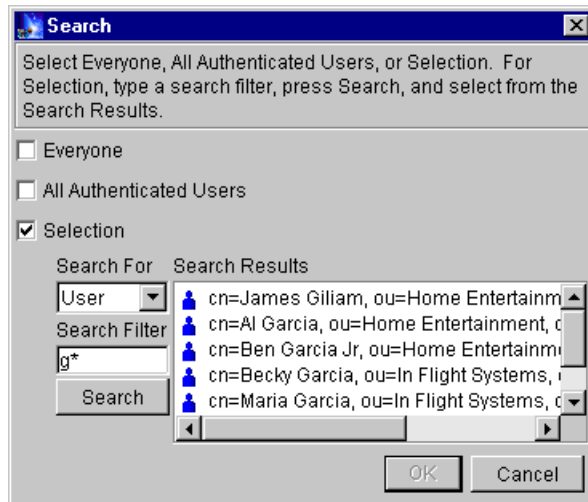


Figure 88. Search dialog for users

We are now done with the WebSphere setup needed to use IBM SecureWay and we will now configure Domino R5 to use IBM SecureWay as well.

6.3.3 Configuring Domino to use the SecureWay directory

We configured Domino to use the SecureWay directory as an external LDAP directory via the Domino Directory Assistance database. The Directory Assistance database allows Domino to look up names and perform Web user authentication from directories in addition to the main Domino directory.

Set up directory assistance like this:

- Start the Domino Administrator client and create a Directory Assistance database on the Domino server using the da50.ntf design template.
Any name can be used for the database and its file name. We used *Directory Assistance* and *da.nsf*, respectively.
- Create a Directory Assistance document in this new database with the settings shown in Figure 89.

Basics	
Domain type:	LDAP
Domain name:	SecureWay
Company name:	IBM
Search order:	
Group expansion:	Yes
Nested group expansion:	Yes
Enabled:	Yes

LDAP Configuration	
Hostname:	freja.lotus.com
Optional Authentication Credential:	
Username:	cn=db2admin
Password:	xxxxxxx
Base DN for search:	o=ibm,c=us
Perform LDAP search for:	<input checked="" type="checkbox"/> Notes Clients/Web Authentication <input type="checkbox"/> LDAP clients
Channel encryption:	None
Port:	389
Timeout:	60 seconds
Maximum number of entries returned:	100

Figure 89. Settings for Directory Assistance document for SecureWay directory

Note that the Domain name and Company name fields in the Basics page can be any names.

- The **Hostname** field should be set to the fully qualified TCP/IP host name of the server running the SecureWay directory. We used: *freja.lotus.com*

- The **Optional Authentication Credential** Username and Password should be set to the directory administrator username and password.

Note: You will be able to see the password in clear text, so you may want to use Notes document-level encryption to protect the contents of these fields. This encrypts the document with a secret key that must be added to the Notes ID files of the server and any administrators who need to access the LDAP configuration document. Other users with access to the Domino Directory, but without the secret key in their Notes ID, cannot view the document. For more information, see the Domino R5 online Administration Help under Directories-> setting up LDAP directories->Encrypting an LDAP directory assistance document.

- The **Base DN for search** field is set according to the directory tree contents. We specified our suffix:

`o=ibm,c=us`

- We set the **Channel encryption** field to *None*, but recommend that SSL channel encryption be used in a production environment.

- Although we set **Perform LDAP search for:** to only Notes Clients/Web Authentication for our testing, you can also check LDAP clients if this is appropriate to your environment.

- In addition to these settings, the **Enabled** and **Trusted for Credentials** fields for Rule1 on the Rules page shown in Figure 90 should be set to *Yes*. This assumes you accept the other default settings for Rule1.

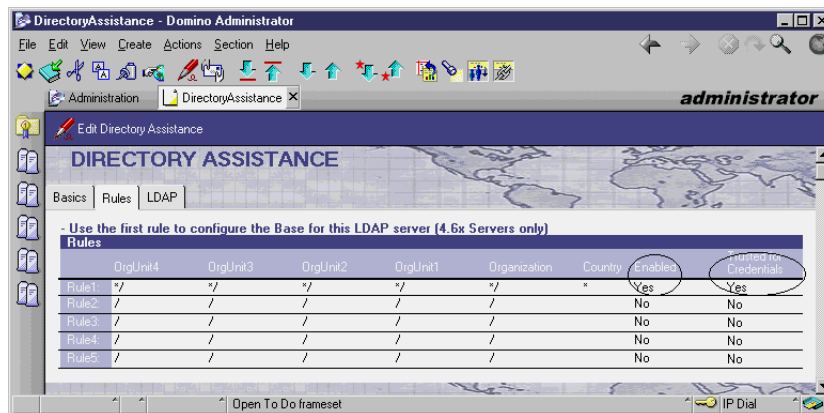


Figure 90. Domino Directory Assistance Rules tab "Trusted for Credentials"

Save and close the Directory Assistance document.

- Next, using the Domino Administrator client, edit the server document for the Domino server. On the Basics tab set the Directory Assistance database name field to the file name of the database you created. For us, this was `da.nsf`. (See Figure 91.)
- Save and close the server document. You can restart the Domino server or wait for the Server Document change to be reread by the Domino server.

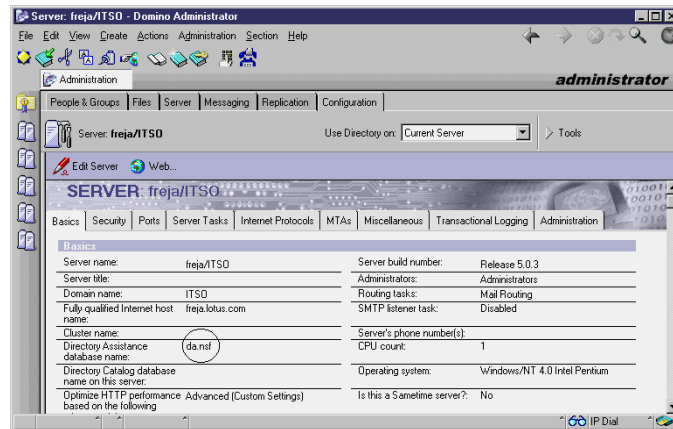


Figure 91. Directory Assistance Database name in Domino Directory Server document

Now users and group names in the SecureWay directory can be added to Domino ACLs to protect Domino databases. Note that the full distinguished name (DN) of users and groups in the SecureWay directory must be specified in the ACL. For example, to add the *Bowling team* group from the sample directory entries to an ACL, specify it as:

```
cn=Bowling team/ou=Groups/o=ibm/c=us
```

(although when displayed in the ACL it will appear as:

```
Bowling team/Groups/ibm/us).
```

We are now ready to use authentication and authorization in our combined Domino/WebSphere application based on basic authentication. Before looking at the application we will review how to set up basic authentication in Domino and WebSphere.

6.4 Using basic authentication in Domino and WebSphere

Basic authentication (where a Web server asks the user to provide a name and password) can be used to authenticate users for access to protected resources on both Domino and WebSphere. Note that basic authentication alone is *not* a secure means of authentication since the password can be easily decoded from the HTTP data sent to the server. The authentication data must be secured by securing the browser-to-server exchange, such as by using Secure Sockets Layer (SSL) encryption.

Let's review the basic authentication protocol since it will help to understand how Domino and WebSphere operate. The basic authentication protocol is defined as part of HTTP. It specifies that a server can request authentication (challenge) in response to any request from a client and requires the user agent (browser) to provide a name and password in the subsequent request header. The server's challenge is made relative to a server-specified realm (a text name) and is only valid for that realm. (A realm can be thought of as the set of URLs with the same root path or, in other words, the same host name and directory as the URL which was challenged.) The user agent may send the name and password in request headers for other URLs in the same realm without being challenged by the server. (In fact, most browsers do this to avoid repeatedly prompting the user for name and password.)

6.4.1 Configuring Domino for basic authentication

Domino uses basic authentication whenever a Web user requests access to a Domino database whose ACL restricts access (that is, has Default and Anonymous entries set to *No Access*) and client-side SSL is not in effect.

The name and password supplied by the user is verified against the Domino directory and any external directories configured via Directory Assistance. Domino can be configured to match various forms of the user name against the directory entry, for example, the common name, the last name, the short name (uid attribute in the LDAP schema) or the full DN (canonical name in Notes terminology).

To set up Domino to use basic authentication for Web access, you must specify that name and password is required for the Web Internet port. Using the Domino R5 administrator client do the following:

1. Select the **Configuration** tab; then select **Server->Current Server Document**.
2. Click the **Ports - Internet Ports** tab; select the **Web** tab.

3. Specify **Yes** for the Authentication options: Name & password field associated with the TCP/IP port (or with the SSL port if SSL is also used for Web access).
4. Save and close the server document, then restart the Domino HTTP server task.

With Domino R5 you can also specify the realm name Domino will return to the browser when it issues challenges for protected resources. (Prior to R5, Domino returned the path of the URL which was challenged as the realm name.)

Using the Domino R5 administrator client, select the **Configuration** tab, then select **Server->Current Server Document**. On the action bar just above the server document view, select the **Web** action (see Figure 92) then the **Create Realm** menu item.

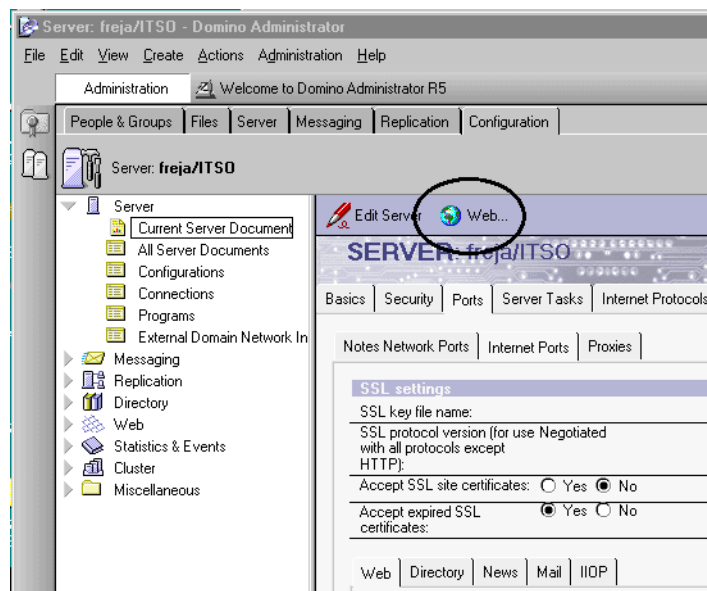


Figure 92. Creating a Web Server configuration

In the Web Realm document (Figure 93 on page 167), specify the local filesystem path of the resources you want to be included in the realm (for example, to include all databases under the Domino data path, specify C:\Lotus\Domino\Data). Then specify the realm name to be returned to the browser. (You do not need to specify the IP Address field unless you are using a Domino virtual host.)

WEB REALM for freja/ITSO	
<div>Basics Administration</div>	
Web Server	
Applies to:	freja/ITSO
IP Address:	
Path	
Path:	d:\Lotus\Domino\Data
Realm returned to browser when access is denied:	lotus.com

Figure 93. Domino Web Realm document

We specified the realm to be the same as we are using in WebSphere:

lotus.com

6.4.2 Configuring WebSphere for basic authentication

WebSphere can be configured to use basic authentication for any or all of the enterprise applications configured in WebSphere. For an enterprise application configured to use basic authentication, WebSphere challenges the user for a name and password whenever an attempt is made to access a protected enterprise application resource.

WebSphere verifies the name and password against the directory set up as the user registry by doing an LDAP bind operation with the given name and password. In our testing, we could successfully authenticate with WebSphere if we specified the name as the value for the *uid* attribute (part of the *inetOrgPerson* object class) or as the full distinguished name (DN) of the directory entry.

Basic authentication can be set as the global default for all enterprise applications on the WebSphere server or it can be specified for any single enterprise application.

To set basic authentication as the global default, use the *Specify Global Settings* task wizard in the Tasks tree in the WebSphere administrative console, as we described earlier in 6.2.2, “Configuring WebSphere to use the Domino Directory” on page 150, or 6.3.2, “Configuring WebSphere to use the SecureWay directory” on page 157. The Application Defaults page in the administrative console where basic authentication is specified is shown in Figure 94 on page 168.

Similarly, basic authentication can be set for an enterprise application. You define an enterprise application using the WebSphere administrative console to define which EJBs, servlets, JSPs and HTML pages comprise the enterprise application. We will walk through this process in 6.5.2.1, “Configuring the BankApp Enterprise Application” on page 171. You specify to use basic authentication for an enterprise application by using the *Configure Application Security* wizard in the administrative console, then choosing the application to configure.

When configuring basic authentication in WebSphere, a realm name must be specified. This is the realm name returned by WebSphere when a challenge for a protected resource is issued. You can assign realm names to all resources on a WebSphere server or per enterprise application. The following section describes how you might use realm names when setting up a Domino/WebSphere application.

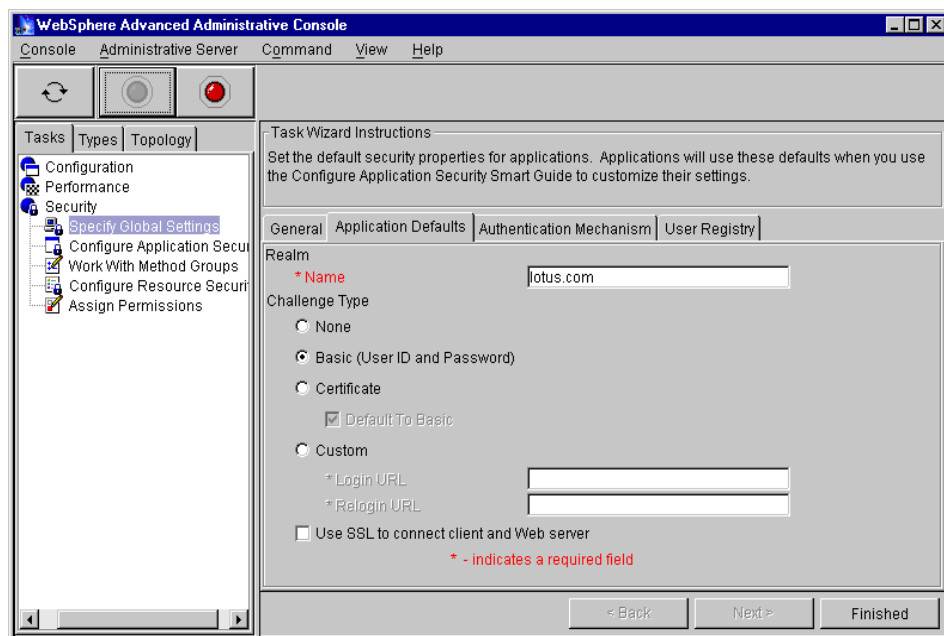


Figure 94. Application defaults for basic authentication.

Note that when basic authentication is configured in WebSphere, then some authentication mechanism must be selected, either Local Operating System or Lightweight Third Party Authentication (LTPA). In the following sections, we discuss basic authentication using LTPA and either the Domino or SecureWay directories as the WebSphere user registry.

Note

In testing, we found that the realm name specified to WebSphere 3.02.1 was not returned on the challenges issued by WebSphere when using the Domino HTTP server due to a problem with the domino5.dll DSAPI plug-in. This problem prevented us from using common realm names to avoid dual authentication prompts.

6.5 Protecting a Web application under Domino and WebSphere

To control Web user access to an application consisting of Domino and WebSphere components, you must take steps to protect the individual components.

- Domino components (databases) are protected by adding entries (user or group names) to the Domino database Access Control Lists (ACLs).
- WebSphere components (HTML pages, servlets, JSPs or EJBs) are protected by configuring them as parts of a WebSphere Enterprise Application (EA) and specifying access permissions for the enterprise application, in other words, who can do what to the application.

Once the components of an application are protected, then user authentication mechanisms can be configured which will guarantee that only the desired users access the application and its functions.

We illustrate how to protect an application made up of both Domino and WebSphere components by working with a simple demonstration application made up of the examples we went through in the earlier chapters. The application is part of a hypothetical Web bank and performs new account setup for registered customers. We call it BankApp. BankApp consists of:

- A Domino database (banking.nsf)
- A servlet (CreateAccountPost)
- An EJB (Account)

The banking.nsf database contains a Notes form (NewAccount) which is designed to invoke the CreateAccountPost servlet upon submission. The CreateAccountPost servlet in turn creates the customer account by invoking the Account EJB. Figure 95 on page 170 shows a diagram of the BankApp application.

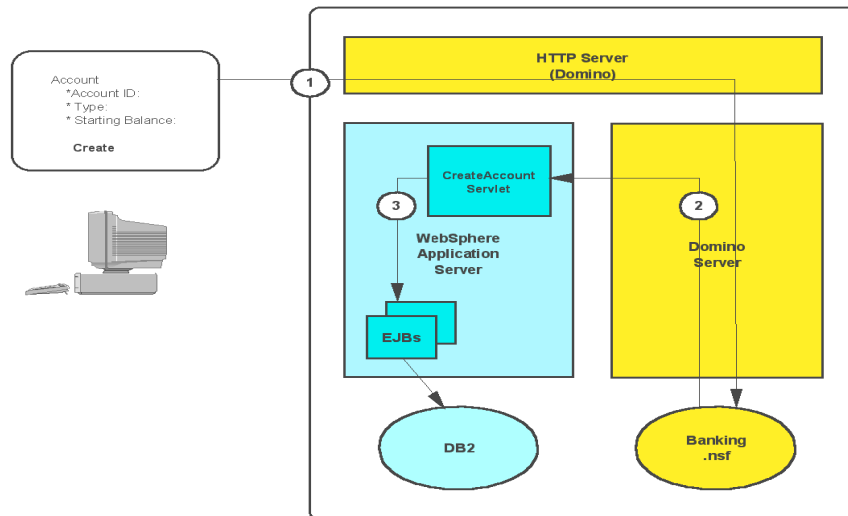


Figure 95. BankApp application diagram.

Since the BankApp application is intended to be used by registered bank customers only, we want to protect the components against access by any user except registered customers. We assume that a user is a registered customer if they exist in the Domino/WebSphere shared directory (user registry) and can be authenticated. (6.1, “Approaches for directory sharing” on page 146 explains how to establish a shared directory among Domino and WebSphere.) Since it is easier to manage user access controls by placing users in groups and granting access to the groups, we will assume that all registered customers are defined to a group named Customers.

6.5.1 Protecting Domino components of a Web application

Domino databases are protected by specifying names in the ACL of the database along with the access rights granted to the named entities. Names of users, groups, servers or special entries such as Default or Anonymous can be specified. A Default entry specifies the rights granted to any user not named or included in any other entry in the ACL. Anonymous specifies the rights granted to any unauthenticated Web (or Notes client) access.

To protect our BankApp application in Domino we did the following to the ACL of the banking.nsf database:

- Added the **Customers** group name with `Editor` access.

- Changed the access rights of the **Default** and **Anonymous** entries to `No Access`.

Note: If you are using IBM SecureWay as the shared directory you may, for testing, want to use the sample group:

`cn=Bowling team/ou=Groups/o=ibm/c=us`

in the Domino database ACL instead of creating a new Customers group.

6.5.2 Protecting WebSphere components of a Web application

Protecting the WebSphere components of an application involves configuring them as a WebSphere Enterprise Application, configuring security for the enterprise application, *and* enabling security (authentication) for the WebSphere server. Note that by default, WebSphere components other than EJBs are *not* protected.

6.5.2.1 Configuring the BankApp Enterprise Application

In order to configure the BankApp as an enterprise application, we first need to install the individual servlet and EJB components. For the Account EJB, we did this in 4.3, “Deploying the IBM WebSphere account example” on page 67. Here is a recap of the steps we followed:

- Set up the DB2 database used by the Account EJB.
- Create the container for the EJB (we named ours Bank Container).
- Create the data source and JDBC driver for the sample DB2 database, associating the data source with Bank Container.
- Deploy the Account EJB.

Using the WebSphere administrative console, we configured a WebSphere *Web application* for the CreateAccountPost servlet. Establishing a Web application allows you to define a virtual path for invoking servlets (or HTML pages or JSPs) and to place the servlet classes in a path anywhere on the file system. To configure our BankApp Web application we did the following:

1. From the administrative console, select the **Tasks** tab and expand the **Configuration** branch of the tree.
2. Select the **Configure a Web application** task wizard and start it.
3. Specify a name for the Web application. We used
`BankApp`
4. Select to **Serve Servlets By Classname**; then click **Next**.
5. Expand the nodes and choose a servlet engine.

We chose ServletEngine under the Default Server, which was the only choice in our configuration.

Click **Finished** and the BankApp Web application is defined.

You can check the configuration by viewing the **Topology** tab.

Note that you will need to create the subdirectories for the paths specified as the Document root and Classpath for the Web application. In our case we created the directories along the path:

```
\WebSphere\AppServer\hosts\default_host\BankApp\servlets\com\ibm\ejb\
doc\sg245955
```

This is where we placed the CreateAccountPost.class servlet class file. You can either copy it from its location under the WebSphere default servlet directory, where we placed it in 5.1.3.3, “Deploying and testing the CreateAccountPost Servlet” on page 115, or you can export it again from VA Java to the servlet directory for our BankApp Web application.

There is now a new relative URL to use to invoke the servlet, which is:

```
/webapp/BankApp/servlet/com.ibm.ejs.doc.sg245955.CreateAccountPost
```

To accommodate that we must modify the Domino form that invokes the servlet using the following steps:

1. Open the database with the Account form we created in 5.1.3, “Posting data to servlets from Domino R5 forms” on page 111 in Domino Designer.
2. Copy the Account form to a form named:

NewAccount

3. Change the Pass-Through HTML form definition to the following:

```
</Form>
<FORM METHOD=POST
ACTION="/webapp/BankApp/servlet/com.ibm.ejs.doc.sg245955.
CreateAccountPost" NAME="_AccountServlet">
```

4. Save the form.

The Domino part of our combined application is now ready.

We will now configure our enterprise application in WebSphere.

- In the WebSphere administrative console, stop the default server.
- On the **Task** tab select the **Configure an enterprise application** task wizard from the **Configuration** tree branch.

- Follow the subtask screens.

We named our enterprise application **BankEnterpriseApp**.

WebSphere applies its security controls to the enterprise application, so it is important to ensure that all of the application components controlled by WebSphere are made part of the enterprise application.

We added the *Account EJB* and the *BankApp Web application* to the BankEnterpriseApp enterprise application. Adding a Web application to the enterprise application will add all of the Web application components (servlets, HTML, JSPs).

Having created the enterprise application we are now ready to configure security for it.

6.5.2.2 Configuring security for the BankApp application

Security controls for a WebSphere enterprise application must be configured separately from the definition of the application, otherwise the enterprise application is unprotected.

In order for certain security settings to be made for an enterprise application, security for the entire WebSphere administrative server must have been previously enabled and a user registry configured. Once you have made sure this is the case, proceed with the following:

- From the WebSphere administrative console **Tasks** tab, use the tasks on the **Security** branch to configure security for an enterprise application.
- First, select **Configure Application Security** to enable security for the entire enterprise application.

After choosing your enterprise application, the task wizard presents a page of settings similar to those on the Application Defaults page of the Global Security settings. You can specify values to be applied to the enterprise application which are different from the global settings, otherwise the global values will be used.

The Global Security settings are used as an initial template for Enterprise Application security. The Enterprise application security could be greater or less than the initial Global Security settings. Changes to the Global Security settings subsequent to setting up an Enterprise Application's security settings will not affect the application's security, with the exception of server global settings such as authentication mechanism.

Click **Finished** when done, *even if no changes to the values are made*.

- Next, select the **Configure Resource Security** task and start the wizard to set security for the components of the enterprise application.

We first configured the Account EJB, accepting the default security settings.

We also configured the webapp/BankApp/servlet resource and accepted the assignment of the default WebSphere method groups.

Click **Finished** to set the configurations.

Note: If you need to configure more than one resource, you *must* click **Finished** after each resource configuration, otherwise the resource security settings you entered will be discarded.

- Finally, select the **Assign Permissions** task to set which users/groups are permitted to invoke which methods of the resource objects—in other words, who is allowed to do what to objects of an enterprise application.

The permissions are assigned to enterprise application/method group pairs. The methods of EJBs and servlets or JSPs (such as HTTP GET, POST) are placed into method groups according to the type of function they perform; for example, read, write, execute.

You can define method groups other than the ones provided by WebSphere by using the **Work With Method Groups** task, but we did not do this.

- To assign the permissions, select one or more enterprise application/method group pairs (hold the Ctrl key to select multiple pairs), then click the **Add** button.

A Search dialog is presented from which you can select all, all authenticated, or only specific users and groups to be permitted to the enterprise application/method groups.

For all the BankEnterpriseApp method group pairs, we assigned the *Customers* group from our user registry.

Note: Our attempts to use the search function with the Domino directory resulted in the dialog becoming unresponsive. We had to close the administrative console DOS session to terminate the dialog. Because of this problem, we were able to select at most the All Authenticated Users option for assigning permissions. WebSphere was able to use the Domino directory for authentication with this setting.

- To ensure that WebSphere recognizes all of the configuration settings discussed here, restart the administrative server and console, then start the Default Server and the enterprise application.

Stopping the administrative server and console can be easily done by right-clicking on the host node in the topology tree and selecting **Restart**.

This will stop and automatically restart the WebSphere server; you will need to restart the Administrative Console manually.

- Once the administrative server and console are restarted, use the **Topology** tab view to check that the security settings are in place.

The enterprise application should appear as a top-level node in the tree and should have the subordinate entries as shown in Figure 96. The components of the Web application you configured should appear under the `default_host` branch and should have the subordinate entries as shown in Figure 96.

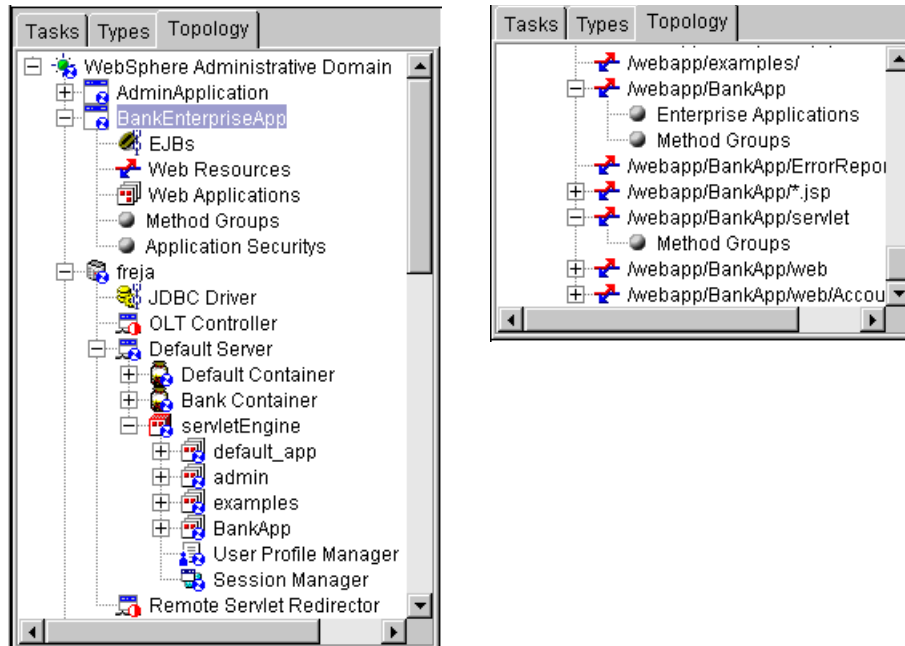


Figure 96. Enterprise application after security configuration

For our BankEnterpriseApp enterprise application, the CreateAccountPost servlet is protected as part of the BankApp Web application.

The Account EJB is protected explicitly as a resource within the enterprise application. The default EJB security settings specify a Run-As Mode of CLIENT, which in our case means that the Account EJB will run under the user who accesses the CreateAccountPost servlet (assuming that the user is successfully authenticated by WebSphere).

Note that we did not make any changes to the CreateAccountPost servlet code, which invokes the Account EJB; the protection was established entirely outside of the application code.

We now have a combined Domino and WebSphere application where access is controlled through all its elements. In the next section we discuss the application behavior when using basic authentication.

6.5.3 Using basic authentication in our enterprise application

With basic authentication configured for both Domino and WebSphere and our BankApp application protected in both, we tested access to our BankApp application. We first accessed the NewAccount form in the Domino database and were challenged for a user name and password by Domino, as shown in Figure 97.

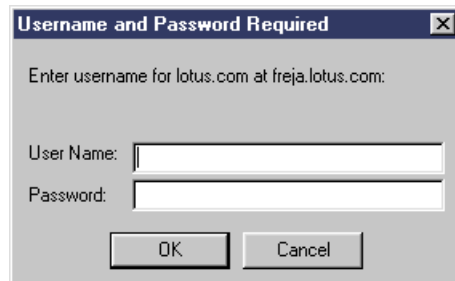


Figure 97. Basic authentication challenge from Domino

Here we were able to specify a user name and password from either the Domino directory or the SecureWay directory depending on which we had configured. (Domino was able to authorize against SecureWay directory even using group names in the database ACL.)

We filled in the Domino form and submitted it. We next received a challenge from WebSphere on the access of the CreateAccount servlet from the form submission; this is shown in Figure 98 on page 177:

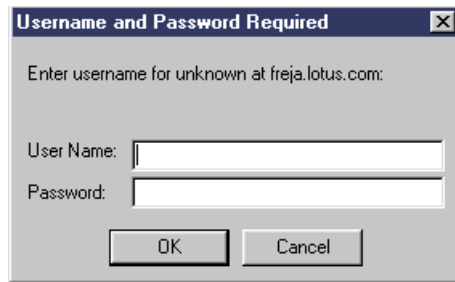


Figure 98. Basic authentication challenge from WebSphere

After re-specifying the user name and password the servlet processing completed, including the call to the Account EJB, without further authentication prompts. When we re-accessed the form after this, we did not receive any authentication prompts since the browser had cached both previous authentications for the respective realms.

Note the setting of the realm name as 'unknown' for the WebSphere authentication; this is indicative of the problem with returning the realm name configured for WebSphere.

When the realm problem in the domino5.dll plug-in is fixed our combined application will be able to share the same realm and we will thus only be prompted for user name and password once, when we access the NewAccount form in Domino.

While we have been using basic authentication in our example, the principles used in securing a common Domino/WebSphere application are the same regardless of what authentication mechanism is used. As mentioned in 1.3, "The technology plan" on page 2, coming versions of Domino and WebSphere will be able to share session-based authentication.

You should be able to develop and test your Domino/WebSphere enterprise application with a setup as we have described in this chapter, and then deploy it on versions of Domino and WebSphere that support shared, session-based authentication with only configuration changes (no codes changes will be necessary).

6.6 Summary

In this chapter we have focussed on authentication and authorization for a combined Domino and WebSphere application. First we discussed the need for a shared directory, and showed how to use either Domino Directory or

another LDAP directory (using IBM SecureWay as an example) as the shared directory for authentication and authorization. We reviewed how to set up basic authentication in Domino and WebSphere. Building on the development work we described in the earlier chapters, we then defined our application and configured it to consist of a Domino database and a WebSphere enterprise application comprising an EJB and a servlet. We secured the different parts of the application and showed that security actually was enforced for all elements of the application.

Chapter 7. Scalability and redundancy with Domino and WebSphere

In this chapter we examine options for scalability and redundancy in a Domino/WebSphere configuration. We discuss:

- The WebSphere Performance Pack products
- Domino Internet Cluster Manager
- Implications of Domino together with the WebSphere Performance Pack

Although this chapter deals with the options for horizontal scaling, which means adding more systems, we should point out that many large enterprises choose to scale their Web sites vertically by moving to large enterprise servers with OS/390. The fact that we do not cover vertical scaling here should not be taken to imply that horizontal scaling is always to be preferred. We cover horizontal scaling in greater detail because there are more complexities to be taken into account when trying to provide a single system image view when your Web site is composed of many smaller machines.

We should also point out that although this chapter deals with configuring for performance, the design and implementation of your application is also of paramount importance. See Appendix E, "Related publications" on page 237 for further reading on designing applications for performance.

7.1 WebSphere Performance Pack

This section gives a brief introduction to WebSphere Performance Pack. It includes:

- IBM SecureWay Network Dispatcher – a load-balancing solution for balancing requests among HTTP, FTP or other TCP-based servers
- IBM Web Traffic Express – a Web caching and proxy server that provides better performance and enhanced proxy solutions
- IBM AFS Enterprise File Systems – a highly scalable file management system that allows you to efficiently manage and non-disruptively replicate files in local or geographically distributed server environments

In the following we will give a brief description of each these three products.

Note: The WebSphere Performance Pack product is being substituted by the *IBM Edge Server* that comprises the functionality of the Network Dispatcher and Web Traffic Express, while IBM AFS Enterprise File Systems is being offered as a product on its own. However, we describe the WebSphere

Performance pack because it is what we had knowledge about at the time of this writing.

7.1.1 Network Dispatcher

The product now known as the Network Dispatcher comes from a prototype developed for load balancing at the Atlanta Olympics Web site.

Earlier load-balancing tools used techniques such as DNS round-robin, but Network Dispatcher can base its decisions on load and availability as well as user-defined criteria such as TCP/IP port number used by the client.

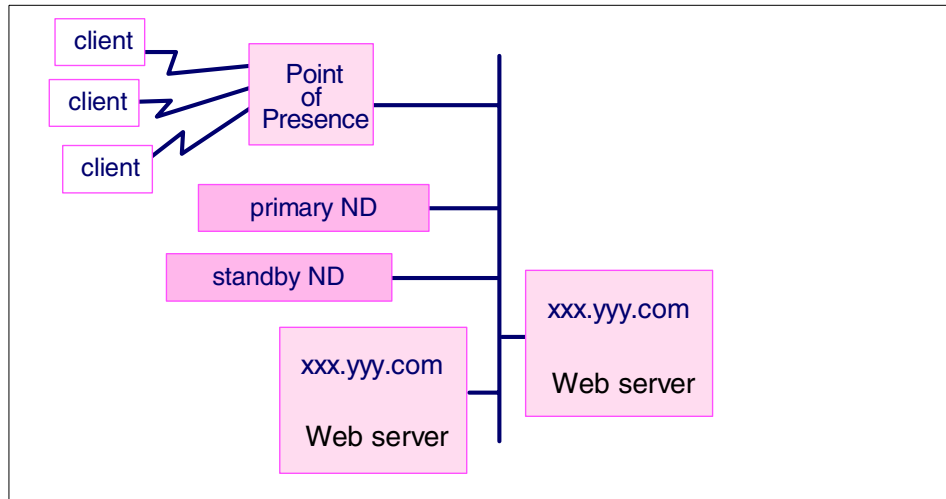


Figure 99. Network Dispatcher

Key advantages of the Network Dispatcher are:

- It does not need agents or any other code to be installed on the systems being balanced.
- It can be configured with a second standby Network Dispatcher to provide availability in the event of the loss of the Network Dispatcher machine itself.
- It does its routing by setting the MAC address in the IP packet header to that of the machine to which it has determined to route the request. This means that all the back-end servers really do have the same IP address and the user receives his reply from the IP address he thought he was talking to.

- Outbound traffic does not go through the Network Dispatcher processor. As inbound traffic volumes tend to be very small compared with outbound traffic, this means that Network Dispatcher can be run on a comparatively small machine and yet handle very high transaction rates.
- The outbound traffic can be routed over a different adapter.

If you are accessing applications on one of the Web servers which require maintenance of session information, Network Dispatcher can be configured to use *sticky port* sessions. That is to say, once the Network Dispatcher has routed a request from a given IP address to a given Web server, subsequent requests from the same IP address, issued within a configurable time-out period, will be routed to the same server.

Refer to the Redbook *IBM WebSphere Performance Pack: Load balancing with IBM SecureWay Network Dispatcher*, SG24-5858 for more information about Network Dispatcher.

7.1.2 Web Traffic Express

The Web Traffic Express (WTE) component of the WebSphere Performance Pack can act as a normal proxy server between a company's intranet and the outside Internet. WTE can also be used in what is known as *reverse proxy* mode, as shown in Figure 100 on page 182 where the actual server being accessed is hidden from the user by WTE.

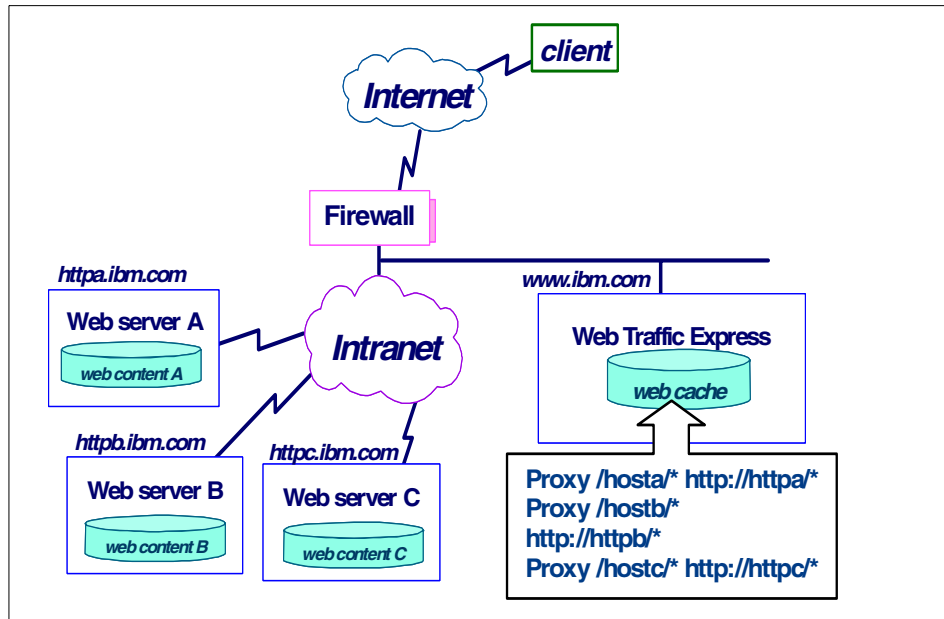


Figure 100. Web Traffic Express Reverse Proxy

With the proxy definitions as shown in the diagram, WTE makes the three Web servers appear as subdirectories of the main *www.ibm.com* server. This has the following advantages:

- The topology of the internal network is hidden from outside users.
- When topology of the internal network is rearranged, a change to the configuration file can prevent the changes from being visible to outside users.
- External users have a *single system image* view of the Web site.
- Pages can be cached at the point of the network closest to outside users.

Domino Usage Notes: You have to take care with the specification of paths to be mapped to different servers because Domino generates references relative to root ("/"). By default, references to icons are generated as, for example:

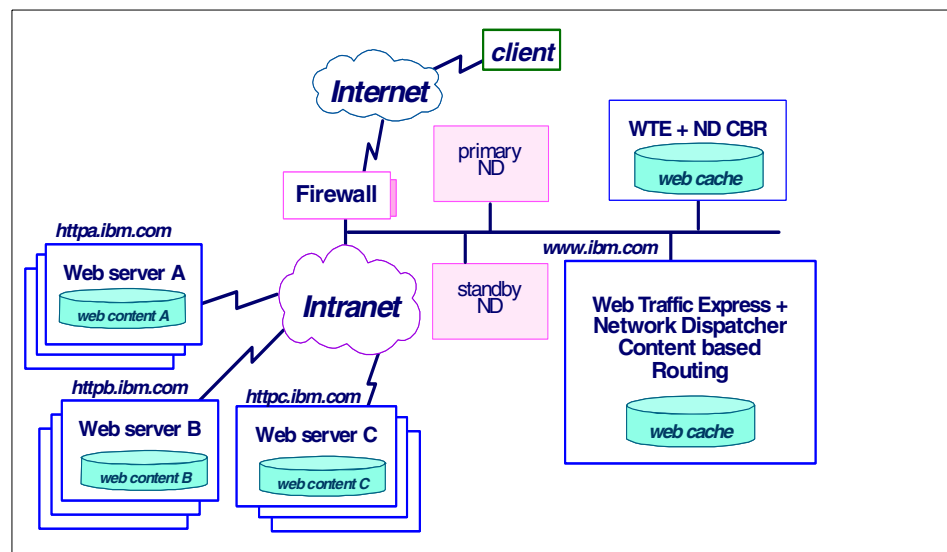
```
<IMG SRC="/icons/vwicon137.gif">
```

Other references are generated as, for example:

```
<A HREF="/dwa/dwa2.nsf/5b7..f7f9!Navigate&To=Prev">
```

If you use WTE as a reverse proxy in front of several Domino servers you will probably also need to include a Proxy statement in the config file mapping server_root ("*/*") to a specific server on which you will at least have a centralized repository for icon files.

7.1.2.1 Content Based Routing



This configuration also gives you the option to use cookies generated by WTE to maintain session affinity with a particular server instead of the *sticky port* setting described above.

accessing your site via proxy servers. Every request accessing your site via a particular proxy server will have the same originating IP address, and via the *sticky port* setting would be assigned to the same back end server. At peak times this could largely frustrate the load-balancing algorithms in Network Dispatcher. The alternative *cookie affinity* setting of Content Based Routing avoids this problem: WTE assigns a cookie to an incoming request and remembers the server to which it routed that request. That cookie will be returned to WTE on subsequent requests from the same browser user and provides a way to distinguish between users accessing the site via the same proxy server.

Refer to the Redbook *IBM WebSphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859 for more information about Web Traffic Express.

7.1.3 AFS enterprise file system

AFS (originally developed at Carnegie-Mellon University and called the *Andrew File System* after Andrew Carnegie-Mellon) is a distributed file system which enables co-operating hosts (clients and servers) to efficiently share file system resources across both local area and wide area networks.

Essentially, AFS provides a single enterprise-wide view of all file servers, with all files having a unique file identification. AFS also caches files near to the point of use.

In a Domino context, AFS is a very useful tool for sharing a single set of those components of a Domino configuration which live in flat files in the file system, rather than in Domino databases which can be replicated.

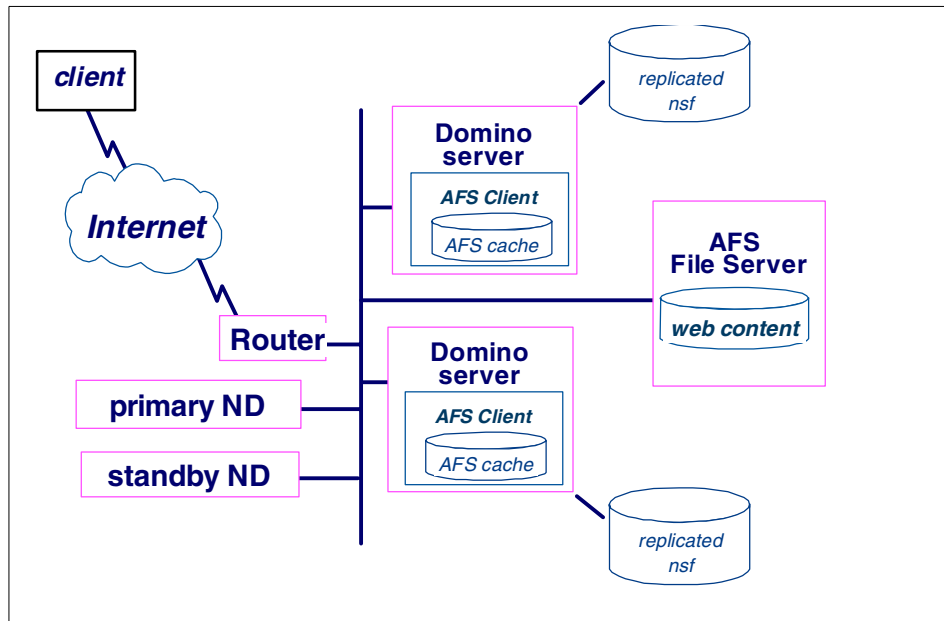


Figure 102. AFS enterprise file system

In a Domino system where you are trying to maintain multiple identical servers for load balancing, the various directories under Domino/data/domino (adm-bin, cgi-bin,html, icons, java) can be kept in “synch” by sharing the files between all servers. Other directories from which html files, images, servlets and JSPs are being served may also need to be “synchronized” using AFS.

Refer to the Redbook *IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System*, SG24-5857 for more information about the AFS file system.

7.2 Domino Internet Cluster Manager

We have now covered the components of the WebSphere Performance pack. Before we discuss how Domino interacts with those components we describe Domino’s own load balancing product: Internet Cluster Manager (ICM).

The Domino Cluster support, as originally implemented, only supported Notes Clients. Domino Release 5 introduced the Domino Internet Cluster Manager (ICM) which extended clustering support to Web browser clients.

The ICM works in the following way:

1. It uses the Domino Cluster Database, which keeps a record of which databases are available on which server.
2. The ICM is a process which can run on a server of its own or can co-reside on a server running the normal Domino HTTP server task. If it co-resides with Domino HTTP, then it has to be configured to use a different TCP/IP port number.
3. The servers participating in the ICM cluster need to know about the ICM server—which they should do if they are part of the same domain and use a common names.nsf file.
4. All incoming browser requests are directed to servers running the ICM.
5. The ICM determines which server(s) hold the requested database and issues an HTTP redirect to the browser, specifying the IP address of an available server which hosts a copy of that database.
6. The browser then fetches the database from that server. Future accesses to other documents in that database go directly to the same Domino server without going through the ICM server.
7. If a link is encountered to other databases on the same server or elsewhere in the cluster, the Domino server generates a link to the ICM so that it can pick an appropriate server to honour the new request.

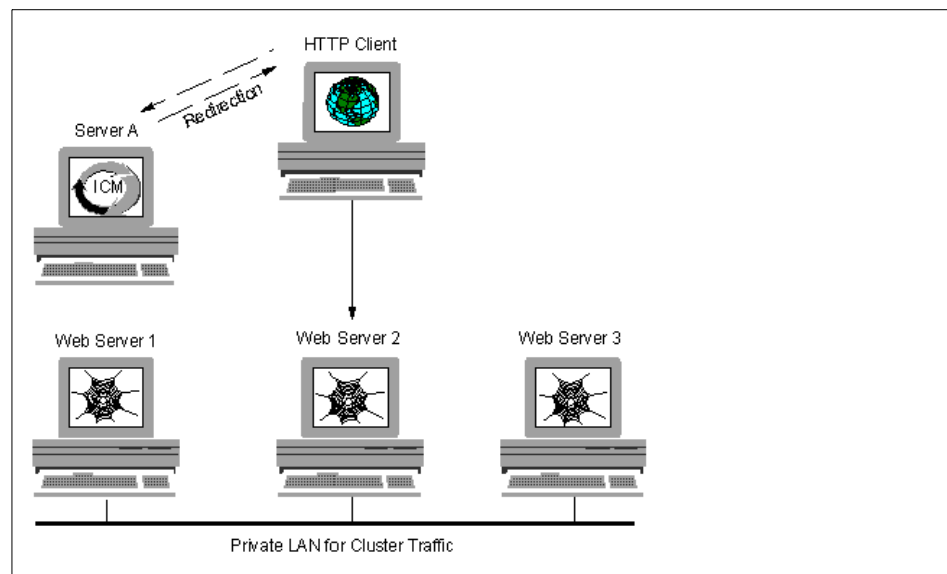


Figure 103. Internet Cluster Manager

Usage Notes: Since a URL redirect is used, if the user looks at the URL of the page he is viewing, he will see the address of the server to which he was directed. Subsequent requests for that database continue to use that server. If that server fails, the user will explicitly have to go back to the ICM to be redirected to one of the surviving servers. If the user bookmarks a page, when he comes back to the site he will go directly to the server he was last using, regardless of its current load or indeed whether it is running at all.

Whether using Basic or Session Authentication, the user will have to log on to each back-end server to which the ICM redirects him.

7.3 Domino authentication and WebSphere Performance Pack

We will discuss how Domino authentication interacts with Network Dispatcher and WebTraffic Express, but first we will review the parts of Domino authentication that are relevant for our ensuing discussion.

7.3.1 Domino authentication reviewed

If you don't set up your Domino server to use X.509 SSL certificates, you have two authentication options for internet users:

- *Basic authentication*, as defined in RFC1945 (and RFC2068 for HTTP/1.1). RFC is an abbreviation for Request For Comments. All Internet standards are documented as RFCs.
- *Session authentication*, which uses the cookies as originally introduced by Netscape, now defined in RFC2109.

If you are thinking about building a site with several Domino servers which contain protected databases, it is important to understand the implications of using the different protection mechanisms.

7.3.1.1 Basic authentication

Basic authentication is the original, and default, method which Domino uses to control access to restricted databases. The protocol works as follows:

1. The first time you try to access a restricted resource, the Web server returns an unauthorized (401) response naming the protected *realm* and the IP address of the server containing the resource.
2. The browser will then prompt you for a user name and password for that realm. You may see a panel which looks like this if you use Internet Explorer:



Figure 104. Basic authentication in Internet Explorer

Here is an example trace of an *unauthorized response* from the server that made the password dialog appear in the browser:

```
HTTP/1.1 401 Unauthorized
Server: Lotus-Domino/Release
Date: Thu, 22 Jun 2000 12:01:59 GMT
Content-Type: text/html; charset=US-ASCII
Content-Length: 297
WWW-Authenticate: Basic realm="/"
```

3. By default, the *realm* is the name of the path containing the file that you were trying to open. Domino R5 allows you to assign your own realm names to specific paths (Open the Server document, then *Create Realm* is an option on the *Web* pull-down).
4. After you enter your name and password, the browser then "encodes" them and resends the get request for the protected resource with the encoded user name and password in the request header.

Here is a sample of what the Web browser sends back:

```
GET /aimetsnames.nsf HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/msword, */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: vampire.isc.uk.ibm.com
```


Proxy-Connection: Keep-Alive
Authorization: Basic RQZ3Xda3DbFjiLG1zbGV5L0FJTTRVWUy9VS==

5. If the user name and password are valid the server returns the requested document.
6. The browser caches the user name and password (in memory) and transmits them in the header of every subsequent request to that domain and realm.

Usage notes: The realm */pathname* is considered to be a subset of the realm */*. This means that if you have authenticated against a resource in the realm */* the browser will expect the same user/password combination to work against realms like */pathname* and will transmit the same user/password without further prompting. On the other hand, if you have first authenticated against the */pathname* realm and then try to access */*, the browser considers this to be a higher level realm and will not pass the cached user/password in the request header. If the resource in the higher level realm turns out to be protected, then you will be prompted again for your user name and password.

However, RFC1945 states *"Proxies must be completely transparent regarding user agent authentication. That is, they must forward the WWW-Authenticate and authorization headers untouched, and must not cache the response to a request containing authorization."*

This implies that if you are using a proxy server and want to make the best use of its cache, you should place read-only reference databases into one realm, and databases which require authentication into another named realm (which is not */*). If you do have an ACL-protected database in the */* realm, the browser will forward the Authorization string with every subsequent request to the server and thus defeat caching.

Note: The *base64 encoding* of the user name and password renders them unreadable to casual users, but the encoding is easily unscrambled. So if you are worried that a hacker could trace the IP traffic, Basic authentication is not very secure, especially since the encoded user name and password are sent in the header of every request for resources in the same realm.

7.3.1.2 Session (cookie) based authentication

Domino R5 introduced the option of using session authentication based on cookies instead of the default Basic authentication.

Session authentication is enabled under the Domino Web Engine tab in the Server document.

Session authentication offers the following advantages over Basic authentication:

- Explicit Login and Logout commands are supported (by appending ?Login or ?Logout to a Domino URL).
- A time-out can be forced after a given period of inactivity.
- A custom-tailored Login page can be provided. As this tailored Login page is a form in a Domino database, you can give it the look and feel appropriate to your Web site and it can be encrypted over SSL to protect the user/password combination from exposure to hackers.

If you set your Web browser to prompt you before accepting cookies, you can display details of the Domino Session cookie as shown in the example from Internet explorer in Figure 105.

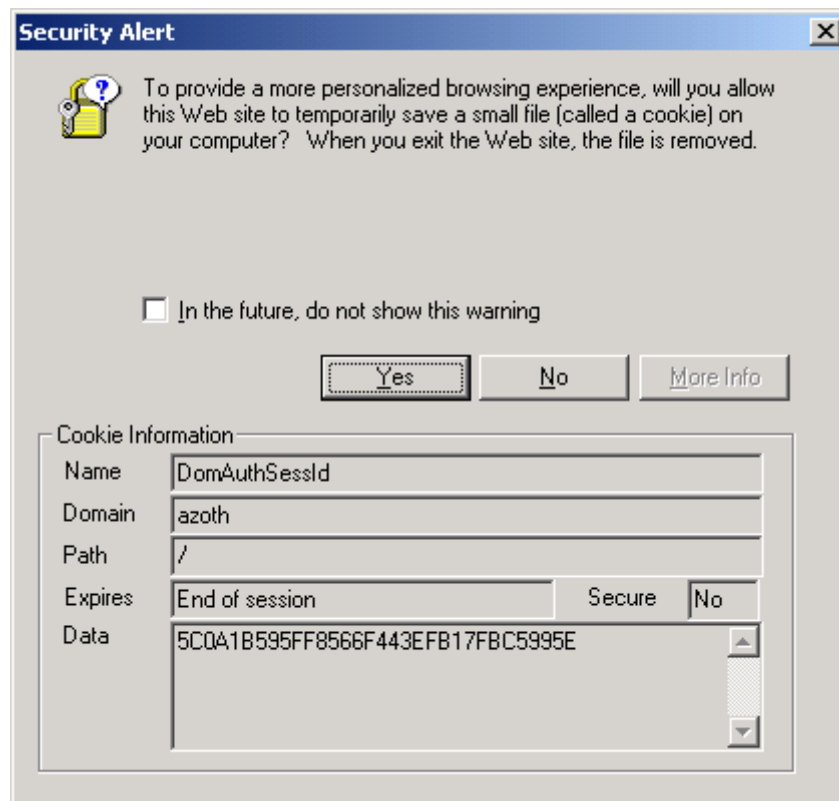


Figure 105. Notification about Domino session cookie in Internet Explorer

Like the authentication header information required for the Basic authentication model, the cookie definition includes Domain and Path matching information.

Here is an example of the response from the server with the cookie:

```
HTTP/1.1 302 Found
Server: Lotus-Domino/5.0.4
Date: Thu, 22 Jun 2000 11:25:53 GMT
Location: http://azoth/aimetsnames.nsf/
Connection: close
Content-Type: text/html
Set-Cookie: DomAuthSessId=5C0A1B595FF8566F443EFB17FBC5995E; path=/
```

Note the server has sent a 302 (relocation) response specifying the location of the protected database originally requested.

The *domain* defaults to the IP name of the server. The protocol defined in RFC 2109 allows the server to set the domain name to include a range of systems. For example, the domain *.isc.uk.ibm.com* (note the leading “.”) would include all servers with a name of the form *xxx.isc.uk.ibm.com*. Domino does not provide an external interface to allow you to set the domain name in the response header.

The browser sends the cookie along with every HTTP request to servers whose name matches the domain definition and where the path in the request matches the path in the cookie definition.

Domino does not provide the ability for you to override the default domain name (it is always the server's fully qualified Domain Name taken from the server document) or the path, which Domino always sets to */*.

We may have gone into more detail than required, but you now are at least well prepared for our ensuing discussion.

7.3.2 Network Dispatcher and Domino

When you have multiple identical Domino servers, fronted by Network Dispatcher, this works well with Basic authentication. This configuration is illustrated in Figure 106 on page 192.

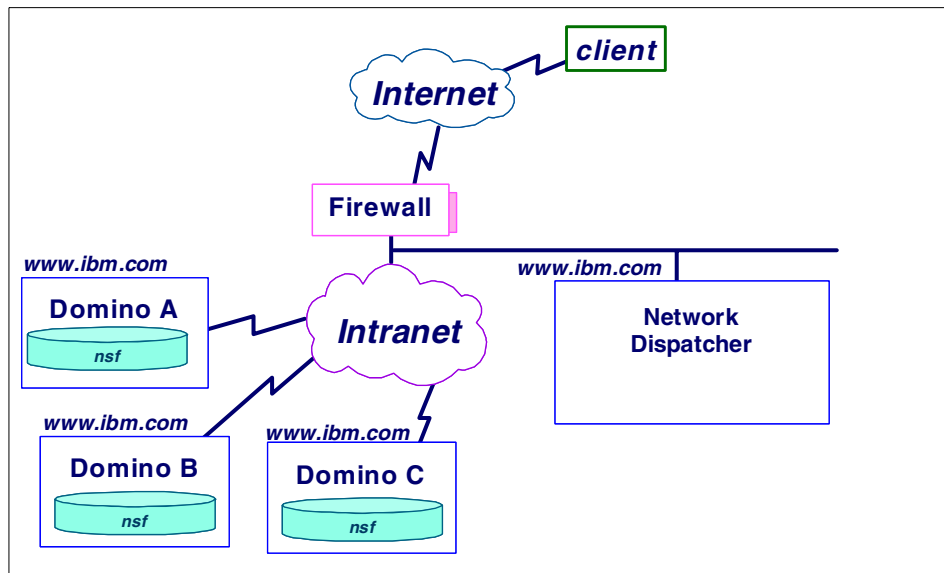


Figure 106. Domino and Network Dispatcher

Once a browser user has authenticated with one of the servers, the browser will pass the validated user name and password on all subsequent requests to any of the servers; since they all have the same IP address the browser is unaware that it is talking to several different servers. If the servers share the same Domino Directory, the same user name/password combination will be valid on each server and the user is only prompted to enter the user name and password once.

Using Domino's session (cookie) authentication, once the user has authenticated against any one server, the browser will return the cookie on every subsequent request to any of the Domino servers (since, as far as the browser knows, they are all the same server with a single IP address).

However, the Domino servers do not have any knowledge of each other's cookies. So when the browser sends back to Domino B the value of the DomAuthSessId cookie issued by Domino A, Domino B does not recognize the cookie. Domino B prompts the user to sign on again and sends a new value of DomAuthSessId back to the browser, overwriting the value set by Domino A.

If Network Dispatcher then routes the user back to Domino A, the user will again be re-prompted to login.

This scenario would clearly be unacceptable.

If you want to load balance across identical Domino servers and to use cookie-based authentication, one option is to use the *sticky port* setting in Network Dispatcher. Requests originating from any given IP address should then always be routed to the same server (except in case of server failure), thus minimizing the likelihood of users being asked to re-login.

Note that WebSphere advanced edition does support persistent cookies (stored in a shared DB2 repository). In a similar configuration multiple WebSphere servers would recognize each other's cookies.

7.3.3 Web Traffic Express and Domino

Just like the scenario with Network Dispatcher, multiple Domino servers behind a WTE reverse proxy cache can be set up to work well with Basic authentication. You should set up realm definitions mapping all the paths to /.

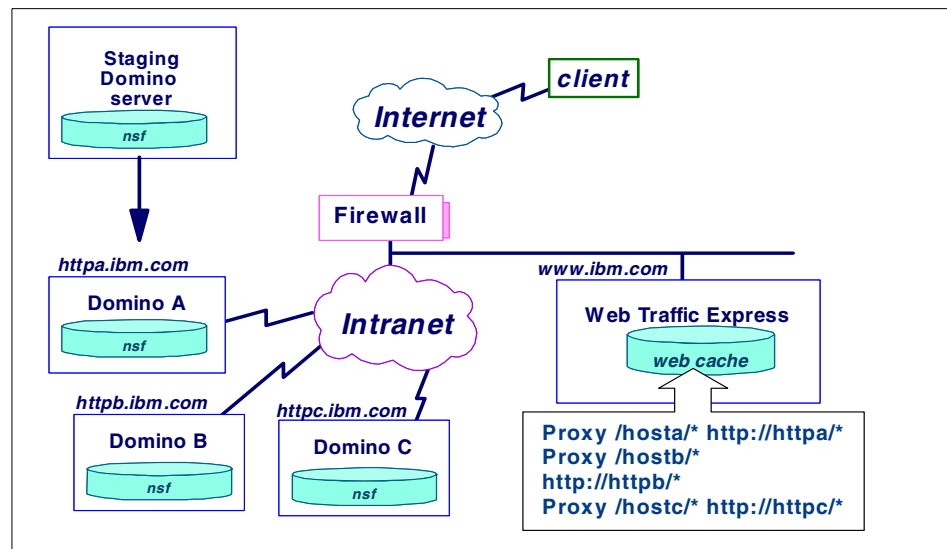


Figure 107. Domino and WTE

Once a browser user has authenticated with one of the servers, the browser will pass the validated user name and password on all subsequent requests; since they all have the same IP address the browser is unaware that it is talking to several different servers. If the servers share the same Domino Directory, the same user name/password combination will be valid on each server and the user is only prompted to enter the user name and password once.

Using Domino's session (cookie) authentication, once the user has authenticated against any one server, the browser will return the cookie on every subsequent request to any of the Domino servers (since as far as the browser knows they are all the same server with a single IP address).

Like the Network Dispatcher scenario, Domino cookie-based authentication will be unusable in this scenario because each Domino server will be assigning its own value to the DomAuthSessId cookie and the browser does not distinguish the servers as separate.

7.3.3.1 Content Based Routing

Web Traffic Express in conjunction with Network Dispatcher can be used to provide Content Based Routing and load balancing across multiple groups of identical back end servers, as shown in Figure 101 on page 183.

Because of the problems with realm names and cookies described above, this will work best with a single group of back-end servers if authentication is being used.

If you decide to use the caching capabilities of the Web Traffic Express component of WebSphere Performance Pack, you have to consider how to make sure that pages which should be cached *are* cached, and that pages which should not be cached are *not* cached.

7.3.3.2 Cache control

According to RFC 2616, by default, a response is cacheable if the requirements of the request method, request header fields, and the response status indicate that it is cacheable.

There are two ways in which the server can issue cache control directives:

- HTTP headers - whereby the server can indicate that a page is not to be cached, or if it is cached, when it should be expired from the cache.
- A <META HTTP-EQUIV="name" CONTENT="content"> tag inserted in the <HEAD> section of the page's HTML.

In the Meta dictionary on <http://vancouver-webpages.com/META/> it is stated that *"While HTTP-EQUIV META tag appears to work properly with Netscape Navigator, other browsers may ignore them, and they are ignored by Web proxies...Use of the equivalent HTTP header, as supported by e.g. Apache server, is more reliable and is recommended wherever possible"*.

HTTP headers can be generated by CGI scripts or Java programs or, with Domino, LotusScript agents. On Domino, the DSAPI can also be used to

overwrite HTTP header information, but this requires programming skills that may not be present in all Domino installations.

Apache and other Web servers also allow HTTP header information to be specified using side files in a directory whose name is defined in the httpd.cnf (httpd.conf) file. The directory name defaults to .Web and the extension for the files defaults to .meta according to the httpd.cnf file that is shipped with Domino on NT. However, although these entries exist in Domino's httpd.cnf file, the Domino server does not honor the definitions.

7.3.3.3 Examples of caching with Domino

We will now illustrate what happens in relation to caching for different Domino Web requests. The tables below shows variations on requests issued for a Domino view and server responses.

In the first example, the browser is calling for a view (called Agenda). If the user has decided he wants to force the browser to reload the page from the server, by using **Control - Refresh**, the browser generates the Pragma: no-cache shown in italics in Table 6.

Table 6. Browser request: Example 1

Browser request
GET /dwa/dwa2.nsf/Agenda HTTP/1.1 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */* Accept-Language: en-gb Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Host: azoth Proxy-Connection: Keep-Alive <i>Pragma: no-cache</i>

The designer of the database created a template for display on the Web, so there is a form named *\$\$ViewTemplate for Agenda*.

If, following the *Expanded Command Caching in Domino 4.61* article in Notes.net, you place a text field on the ViewTemplate form with the name *\$CacheOptions* and set its value to "0", then the server generates an expiry date in the past and the no-cache instruction as shown in italics in Table 7 on page 196.

Table 7. Server response: Example 1

Server response
HTTP/1.1 200 OK Server: Lotus-Domino/5.0.4 Date: Mon, 19 Jun 2000 19:45:27 GMT Connection: close Content-Type: text/html; charset=US-ASCII Content-Length: 15236 Expires: Tue, 01 Jan 1980 06:00:00 GMT Cache-control: no-cache

In the second example we show what happens when the browser is doing a conditional GET for one of Domino's icons.

Table 8. Browser request: Example 2

Browser request
GET /icons/dblsort.gif HTTP/1.1 Accept: */* Referer: http://azoth/dwa/dwa2.nsf/Agenda Accept-Language: en-gb Accept-Encoding: gzip, deflate If-Modified-Since: Wed, 18 Nov 1998 15:07:16 GMT User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Host: azoth Proxy-Connection: Keep-Alive Cookie: DomAuthSessId=195F38B593C256FF8B8C7F80C3468669

Table 9. Server response: Example 2

Server response
HTTP/1.1 304 Not modified Server: Lotus-Domino/5.0.4 Date: Mon, 19 Jun 2000 19:44:31 GMT Connection: close Content-Type: image/gif Last-Modified: Wed, 18 Nov 1998 15:07:16 GMT

You will see from the 304 response that the icon has not been modified, no data is returned, and the browser will use its cached version of the GIF.

For files in the file system, Domino returns the timestamp of the file as the Last-Modified time. Note also the cookie which shows that the user has signed on.

How frequently the browser checks for updated files depends what the user has set. These traces were made with the settings shown in Figure 108 (not necessarily the most efficient).

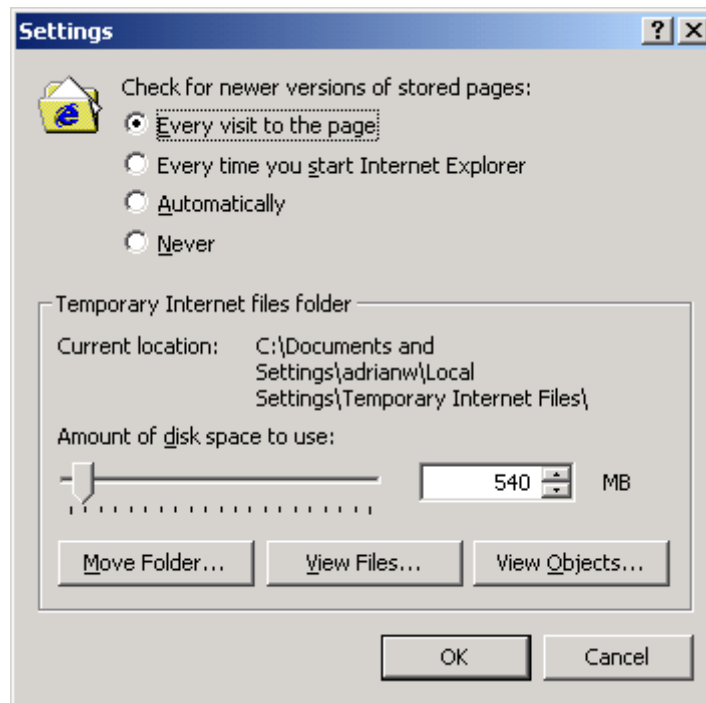


Figure 108. Cache settings for Internet Explorer

In the third example, the browser is doing a conditional GET for one of the image resources used by the Agenda view. The image resource has not been modified, but for any of the contents of a database, Domino sets the Last Modified timestamp to be the latest time when anything in the database was changed. This really means that the most efficient caching of Domino databases is for completely read-only databases. Otherwise, when anything in the database changes, browsers (and caching proxies) will reload everything they have cached.

Table 10. Browser request: Example 3

Browser request
GET /dwa/dwa2.nsf/R5.gif!OpenImageResource HTTP/1.1 Accept: */* Referer: http://azoth/dwa/dwa2.nsf/Agenda Accept-Language: en-gb Accept-Encoding: gzip, deflate If-Modified-Since: Mon, 19 Jun 2000 17:51:44 GMT User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Host: azoth Proxy-Connection: Keep-Alive Cookie: DomAuthSessId=195F38B593C256FF8B8C7F80C3468669

Table 11. Server response: Example 3

Server response
HTTP/1.1 200 OK Server: Lotus-Domino/5.0.4 Date: Mon, 19 Jun 2000 19:45:27 GMT Connection: close Content-Type: image/gif Content-Length: 8134 Last-Modified: Mon, 19 Jun 2000 19:43:50 GMT

7.4 Caching implications of Domino Views and URLs

We now look at how the construction of Domino URLs can effect caching, and also discuss use of the Domino R5 view applet from a caching perspective.

7.4.0.1 “?” or “!” in URLs

Companies with Domino-based servers used to remark that the Web crawlers such as AltaVista and HotBot did not index their sites very well, if at all. This was because they assumed that the “?” in the middle of Domino-generated URLs represented a variable response to a parameterized request and accordingly those pages were not indexed by the crawlers.

This problem was addressed by a new parameter in Domino R5. If you scroll down on the Domino Web Engine tab of the Server document you will find an entry which says *Make this site accessible to Web crawlers*.

When you set this option to *Enabled*, Domino will use a ! Instead of a ? in the URLs which it generates. Note that you cannot apply this setting selectively - it applies to all databases hosted by the server. This option only controls the URLs which Domino generates. If you like typing in Domino’s long URLs by

hand, you can use ? or ! regardless of the setting in the server configuration document.

The presence of the ? also, by default, prevents the WTE proxy cache from caching such pages, whereas the ! does not. However, an investigation of the server trace as shown in Table 12 and Table 13 reveals that many times Domino does not associate Last-Modified times with such pages, which would also tend to inhibit caching.

Table 12. Browser request for a view item

Browser request for a view item
GET /dwa/dwa2.nsf/5b738e664dfa103802566af00799d61/fc7872af60bf8fb88025682e0052d45d!OpenDocument HTTP/1.1 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */* Referer: http://azoth/dwa/dwa2.nsf/htmlmedia/agenda.html Accept-Language: en-gb Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Host: azoth Proxy-Connection: Keep-Alive

Table 13. Server response for a view item request

Server response
HTTP/1.1 200 OK Server: Lotus-Domino/5.0.4 Date: Thu, 22 Jun 2000 13:49:42 GMT Connection: close Content-Type: text/html; charset=US-ASCII Content-Length: 4501

GIFs, however, can be cached, as shown in Table 14 and Table 15 on page 200.

Table 14. Browser request for an image

Browser request for an element of a form (an image)
GET /dwa/dwa2.nsf/4cb99b54b366ccbb802566af006f68a9/\$Body/0.E5A!OpenElement&FieldElemFormat=gif HTTP/1.1 Accept: */* Referer: http://azoth/dwa/dwa2.nsf/5b738e664dffa103802566af00799d61/fc7872af60bf8fb88025682e0052d45d!OpenDocument Accept-Language: en-gb Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Host: azoth Proxy-Connection: Keep-Alive

Table 15. Server response which includes Last-Modified

Server response which includes Last-Modified
HTTP/1.1 200 OK Server: Lotus-Domino/5.0.4 Date: Thu, 22 Jun 2000 16:48:34 GMT Connection: close Content-Type: image/gif Content-Length: 1073 Last-Modified: Mon, 19 Jun 2000 19:43:50 GMT 1073 Bytes Binary Data

7.4.0.2 R5 View Applets and caching

If, using Domino Designer, you set the properties of a view to use a Java applet in the browser, there are several advantages. Scrolling up and down the view and expanding twisties is handled within the applet without interacting with the server. The java applet itself remains resident once loaded by the JVM and the code itself (/domjava/nvapplet.cab for MS IE) can be cached across invocations of the browser.

However, there are two other implications that potentially affect performance:

- The view applet does not respect the *Make site accessible to Web crawlers* setting and unconditionally uses ?s in the URLs which it generates.
- The applet uses a POST with the ReadViewEntries option to retrieve the contents of the view. This means that if, having opened a document from the view, you use the back button to go back to the view, the browser will

always re-issue the POST to retrieve the view contents. This is illustrated in Table 16 and Table 17. For a large view, this could lead to a noticeably delayed response. If this is a problem, you could either not use the view applet or you could arrange that clicking on a link in the view opened up a new browser window. Then you could return to the view applet in its own window without having to use the back button.

Table 16. POST request issued by view applet

POST request issued by view applet
POST /dwa/dwa2.nsf/F195E3299E42CFBE8025679A004B65C2?ReadViewEntries&PreFormat&Start=1&Navigate=15&Count=40&SkipNavigate=0&SkipCount=0 HTTP/1.1 Accept-Language: en Content-Type: application/x-www-form-urlencoded Content-Length: 16 Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2 User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Win32) Host: azoth Proxy-Connection: Keep-Alive Pragma: no-cache

Table 17. Server response header

Server response header
HTTP/1.1 200 OK Server: Lotus-Domino/5.0.4 Date: Thu, 22 Jun 2000 17:10:28 GMT Connection: close Content-Type: text/xml Content-Length: 16165

7.5 Recommendations

Based upon the issues we have covered in this chapter we offer the following recommendations for using Domino together (or perhaps not together) with the WebSphere Performance Pack:

Table 18. Recommendations

If you have	Then
Multiple identical Domino servers	Choose Network Dispatcher with WTE and Content Based Routing and <i>cookie affinity</i> .

If you have	Then
Multiple different Domino servers	Choose Domino's Internet Cluster Manager - although you have to log on to each server, each will recognize its own cookies.
Concerns about interactions with proxy caches	Select a tracing tool and test all major browsers with whatever cache settings you think users may have used. Preferably also test via major ISPs (AOL, Demon etc) to assess the impact of the ISPs' proxy caches.

7.6 A note about tools and techniques

Even though not directly related to our topic, we include here a bit of information on tools and techniques we find useful when working with Domino and WebSphere e-business solutions.

7.6.0.1 HTTP tracing tools

In order to understand the commands issued by browsers and HTTP servers, and in particular the commands which affect caching, you need a tool which can display all the headers sent in each direction. Among them are:

- HttpTracer from Superior Software Tools at <http://www.concentric.net/~Sstmail/index.html> (\$25 Shareware)
- Pluxy (Beta-4) from <http://webtools.dyade.fr/pluxy/>
- Microsoft's Web Application Stress Tool (WAS) <http://webtool.rte.microsoft.com/> - currently downloadable without charge. This is primarily a stress test tool which can record all commands and headers sent by the browser and then replay sessions. It does not allow you to analyze the contents of the HTTP headers sent by the server.

7.6.0.2 Sample DSAPI code for HTTP headers

A good write-up of the Domino caching situation and downloadable DSAPI sample code can be found on the Original Business Systems site at:

http://www.obs.co.uk/OBS/OBSWebPg.nsf/docs/http_headers

We have not evaluated the sample code in the course of the project to write this book.

7.7 Summary

In this chapter we have given a brief overview of the WebSphere Performance Pack and Domino's solution for load balancing: Internet Cluster Manager. We discussed authentication and caching considerations when Domino works together with WebSphere Performance Pack and finally given some recommendations about how to proceed to find the right configuration based on your requirements.

Appendix A. Using the IIS Web Server for Domino and WebSphere

In this appendix we walk through the steps to install Microsoft IIS 4.0 to act as HTTP server for Domino and WebSphere.

If you don't want to work with IIS you can skip this appendix.

A.1 Installation of IIS 4.0

We installed IIS from the Windows NT Option Pack CD-ROM. The CD automatically starts the installation program when inserted in the computer's CD-ROM drive. The initial installation panel (shown in Internet Explorer) is shown in Figure 109.



Figure 109. Windows NT 4.0 Option Pack initial installation screen

To continue, select **Install** from the initial panel and select the component to be installed. To install Internet Information Server, the Option Pack (menu item 5) must be installed. This is shown in Figure 110 on page 206.

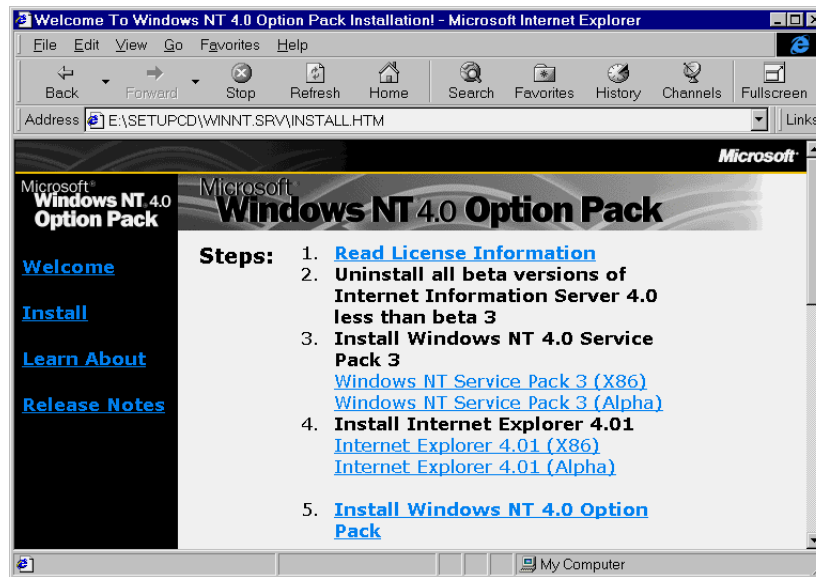


Figure 110. Windows NT 4.0 Option Pack installation options

Once menu item 5 has been selected, it is possible to select which product should be installed from the option pack. This is shown in Figure 111 on page 207. We used the default installation selection, which installs several products in addition to IIS. Note that it is essential to install the Microsoft Management Console (this is the default) since this product is necessary to configure IIS, once installed.

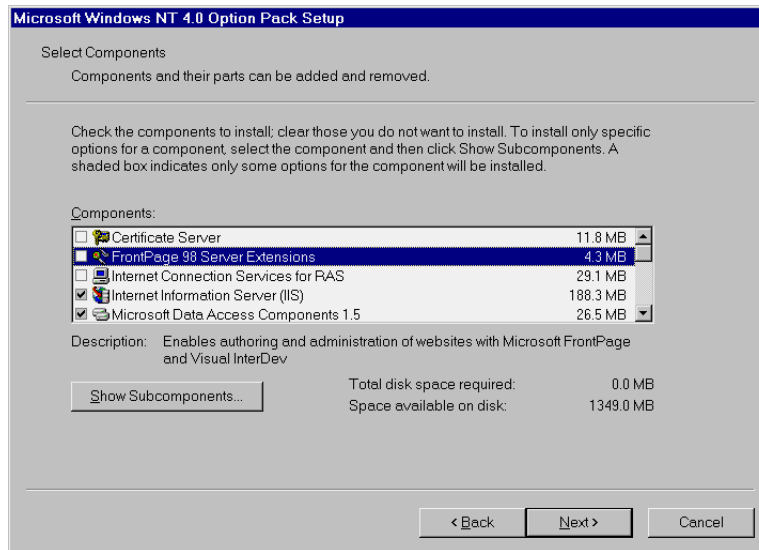


Figure 111. Windows NT 4.0 Option Pack product installation selection

Click **Next** from this panel. The installation will run and complete without further intervention. As usual, it is necessary to reboot the NT Server before using the products.

A.1.1 Configuration of Internet Information Server

Select **Start->Programs->Windows NT4.0 Option Pack->Microsoft Internet Information Server->Internet Service Manager** to start the Microsoft Management Console. The initial console panel shows the Internet Information Server and associated services as illustrated in Figure 112 on page 208.

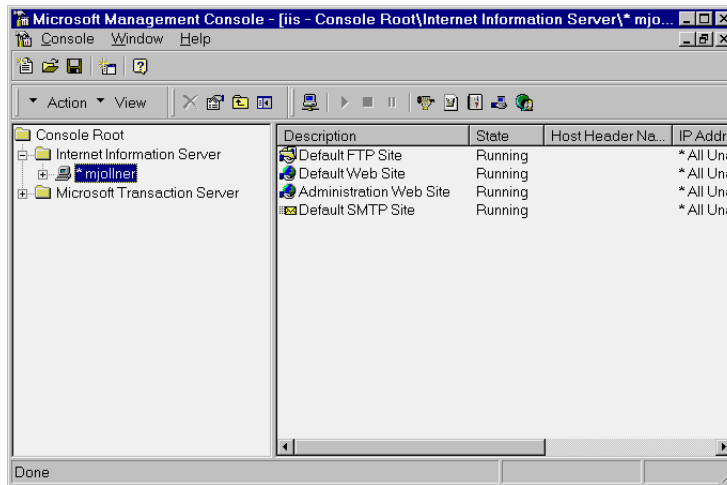


Figure 112. Microsoft Management Console for IIS

Next, right-click on the server (*mjollner* in our example) in the left pane and select **Properties** from the context menu. This will open a new window where you can configure filters and plug-ins for IIS, as shown in Figure 113.

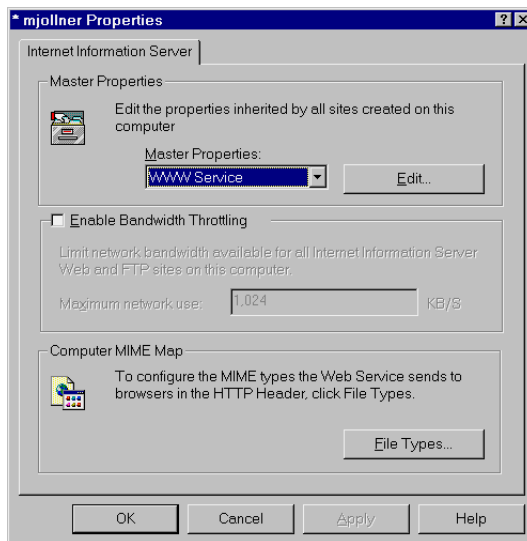


Figure 113. IIS server properties

From the properties panel, click the **Edit** button beside Master Properties - WWW Service. The resulting Master Properties panel, shown in Figure 114, has multiple tabs which allow selection of properties to be configured for IIS.

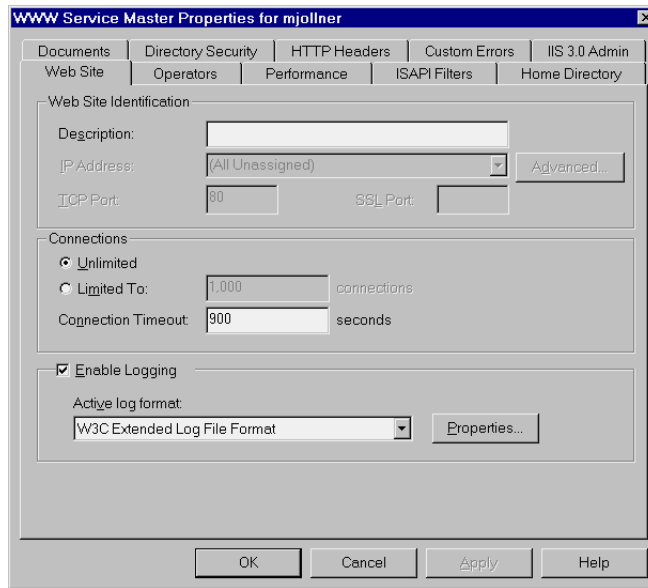


Figure 114. IIS Master Properties panel

A.1.1.1 Configuring IIS for Domino

This description is a brief overview of the installation process: for a more detailed discussion, see the Domino 5 Administration Help File under Web and News Servers, Domino for Microsoft IIS.

To configure IIS to pass user credentials to Domino, select **ISAPI filters** and click **Add** from the resulting panel. Fill in the dialog as shown in Figure 115.

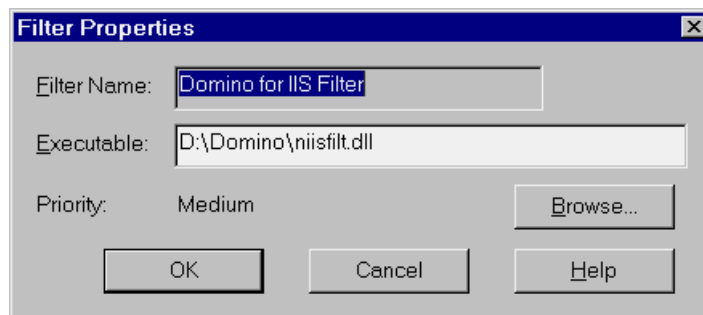


Figure 115. Adding the Domino filter to IIS

You should substitute “D:\Domino\niisflt.dll” with the path to your Domino executables directory. You can also browse to your Domino executable

directory and select **nissflt.dll** from the file list. Click **OK** to confirm your choice.

When the plug-in is installed and loaded, the original configuration panel will show it in the list of loaded plug-ins. This is illustrated in Figure 116, which also shows the WebSphere extension filter (sePlugins) loaded. The installation of this plug-in is shown in Figure 122 on page 215.

Note that the plug-ins may or may not show as loaded after initial installation. This is not an error: they will show as loaded once you access IIS to load Domino (and/or WebSphere resources).

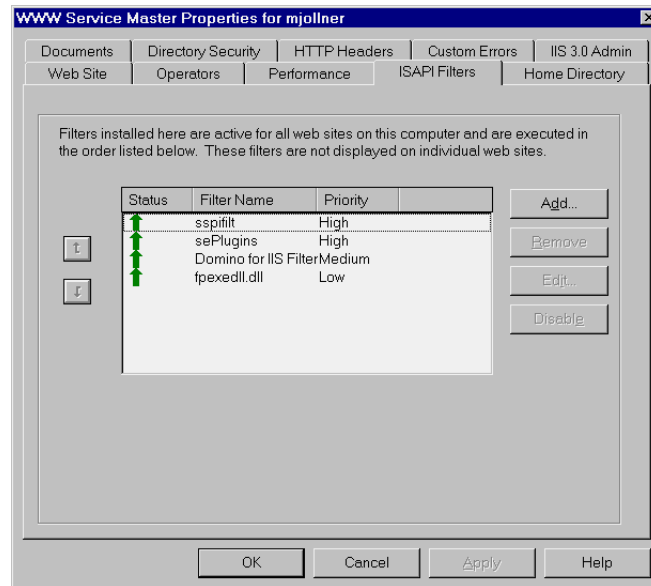


Figure 116. Installed IIS extension filters

To configure IIS to recognize that “.nsf” extensions should be handled by Domino, select the **Home Directory** tab from the Master Properties panel shown in Figure 114 on page 209. Click the **Configuration** button, then click **Add** from the resulting dialog. Enter the Domino ISAPI extension as shown in Figure 117 on page 211, or browse to your Domino executable directory (*d:\Domino* in our example) and select **niisextn.dll**. Click **OK**.

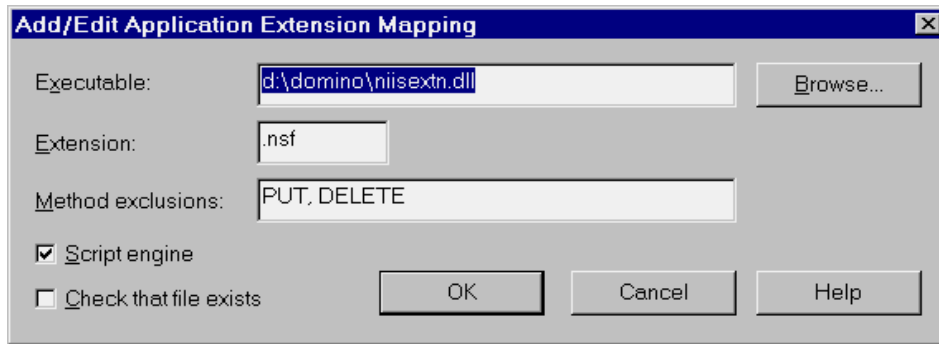


Figure 117. Adding the Domino ISAPI extension

Again, substitute your own path to the Domino program executables as necessary. The Domino filter will be included in the list of Application Mappings as shown in Figure 118. The line containing the Domino ISAPI extension is highlighted.

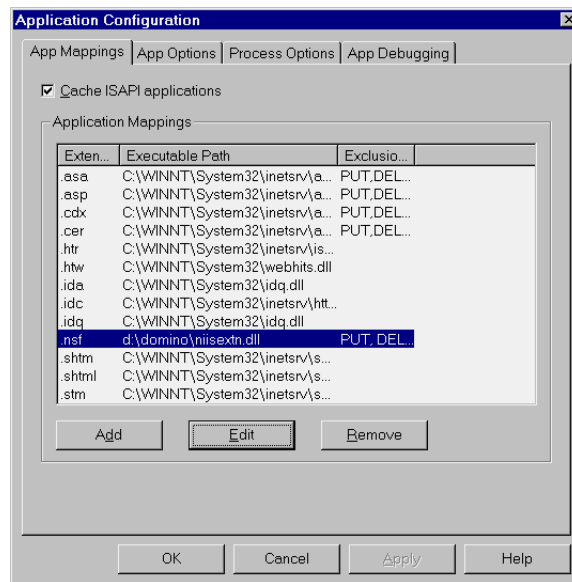


Figure 118. List of ISAPI extensions and Application Mappings

Return to the Microsoft Management Console screen by pressing **OK** as many times as necessary to dismiss the Master Properties dialog.

1. Click the "+" sign to the left of the host name (*mjollner* in our example) under the Internet Information Server folder in the left pane. From the

resulting tree, right-click the **Default Web Site** and select **New->Virtual Directory**.

2. Enter **icons** in the field “Alias to be used to access virtual directory” and then the full path to the Domino http icons. This was *D:\domino\data\domino\icons* in our installation and should be changed to fit your configuration.
3. Click **Next** and then **Finish** on the succeeding panel.

This adds the Domino http icons as a virtual directory to IIS so they can be displayed correctly by Domino applications.

Similarly, add the Domino Java applets by repeating the above procedure, specifying **domjava** as an alias name and the full path to the Domino java applet directory. Again, in our example, this was *D:\domino\data\domino\java*; you will want to adjust it to point to your installation directory. Further details can be found in the Domino 5 Administration Help.

Check the virtual directory to confirm that the setup has been done correctly. Expand the tree under the **Default Web Server** in the left pane of the MMC, then right-click **Properties**. This will show a panel similar to Figure 119, from which the directory settings can be checked and, if necessary, changed. Note that the detailed settings are not important for our testing purposes.

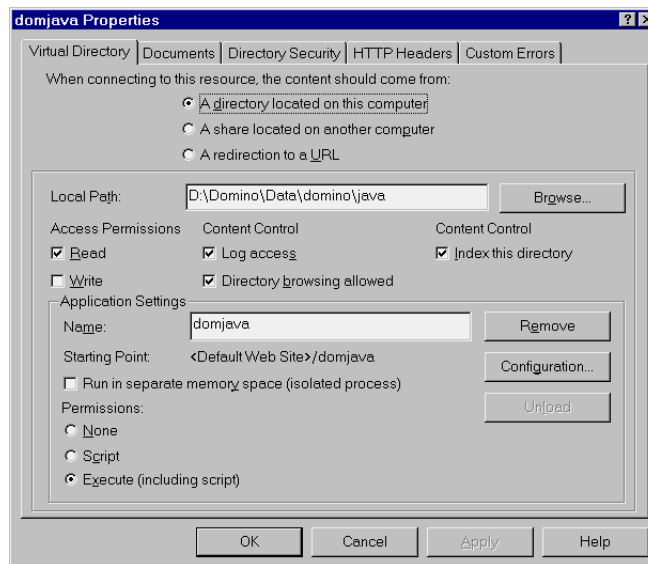


Figure 119. Viewing an IIS Virtual Directory's properties

Note that this procedure has not adjusted the security IIS will apply for Domino. We set this to allow Anonymous and Basic Authentication from the Master Properties dialog (Directory Security tab). This will allow Domino to enforce its own authentication and access control.

You should now be able to test IIS and Domino together. Start the IIS server and, ensuring that the Domino HTTP task is not configured to start, specify a Domino URL (specifying your server), such as:

`http://mjollner.lotus.com/homepage.nsf?Open`

You should see the Domino homepage shown in Figure 120.



Figure 120. Home page displayed by Domino

Note that it is not necessary to have the Domino server running; if it is not, the necessary components will be loaded since Domino can operate as a COM server.

A.1.1.2 Applying the IIS servlet fix to WebSphere

For WebSphere servlet support to work with IIS you need to install a fix on top of the service upgrade we installed in 2.7.2, "Installing the WebSphere 3.02 service upgrade" on page 34. The fix needed is PTF PQ37562. You find it on the same Web site as the service upgrade under the Service Pack and E-fixes section. You may have to register at the Web site to get the fix.

The fix is in a zipped file names *PQ37562.zip*. It contains a new version of the file named *iis20.dll*. You must stop Domino, WebSphere and IIS and copy the *iis20.dll* file to the WebSphere bin directory. In our case the full path is:

D:\WebSphere\AppServer\bin

Note: This fix will be included in the next service upgrade for WebSphere, so if the service upgrade you installed upgrades WebSphere to a higher version than V3.02.1 you should not need to install this fix.

A.1.1.3 Configuring IIS for WebSphere

You configure IIS for WebSphere by a utility program supplied with WebSphere. This program is installed together with the actual plug-in file if you select IIS 4.0 as Web server during the installation of WebSphere.

Using the Windows NT Explorer program, navigate to the WebSphere bin directory. By default, this will be in the \WebSphere\AppServer\bin directory on the drive where you installed WebSphere. It is also possible to simply open a command prompt in the bin directory and run the configuration program. We illustrate the **IIS40cfg** program selected in Windows NT Explorer in Figure 121.

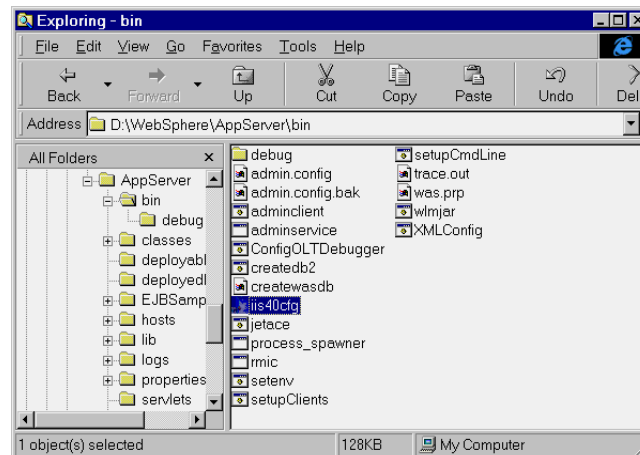


Figure 121. Selecting the WebSphere IIS40 configuration program

Double-click the program icon to start it. This program has a single panel, shown in Figure 122 on page 215.

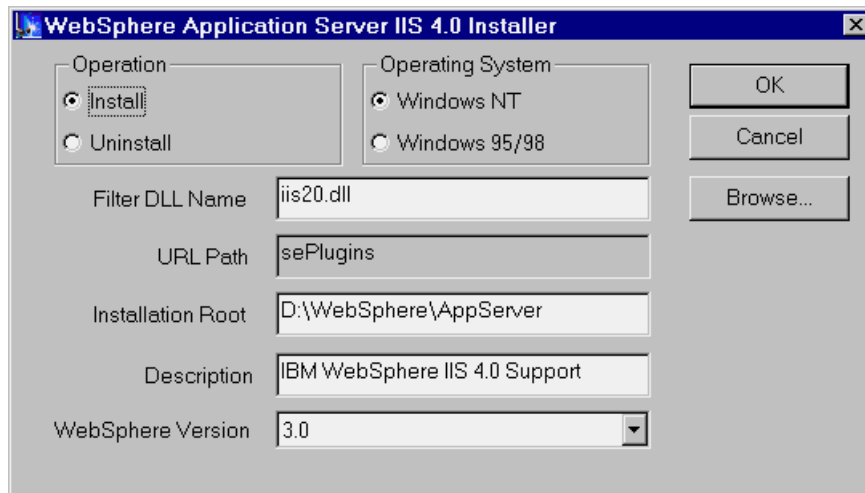


Figure 122. WebSphere IIS 4.0 ISAPI installation program dialog

The defaults in the panel should be correct; if not, correct them as shown and click **OK**. The result is simply an information box which confirms successful installation. Confirmation that the filter has been installed can be checked by loading the *snoop* servlet, as we did for the Domino filter as shown in Figure 116 on page 210.

Note that WebSphere also adds a Virtual Directory *IBMWebAS* with a subdirectory of *Samples*. In the default configuration the Samples subdirectory has an incorrect path to d:\WebSphere\AppServer\Samples; this should be changed to d:\WebSphere\AppServer\WebSphereSamples for the default installation (to a d: drive in this case) if browsing to the WebSphere samples is desired.

We tested connectivity to WebSphere as described previously in 2.7.4.1, “Verifying servlet support” on page 42, with the same results. Installation of the service upgrade to raise WebSphere V3.02 to V3.021 allows IIS and WebSphere to operate together without error, provided the fix PQ37562

(which simply replaces the iis20.dll installed in the procedure shown in Figure 122 on page 215) is installed.

This completes the installation of IIS as HTTP server for Domino and WebSphere.

Appendix B. WebSphere plug-in behavior and tracing options

This section is optional, but may be of interest to understand the connectivity between the WebSphere Application Server and the two HTTP servers (Domino R5 and IIS 4.0) we tested. We also describe some of the options available to trace WebSphere and Domino

B.1 WebSphere HTTP server plug-in behavior

We tested three topologies:

1. Domino R5 (using the DSAPI plug-in) and WebSphere

In this configuration we found that *all* HTTP requests sent to Domino were forwarded through the DSAPI plug-in to WebSphere; WebSphere made the decision which requests to handle itself, returning all others (for Domino objects and html files) for handling by the Domino R5 HTTP stack.

This is shown in Figure 123.

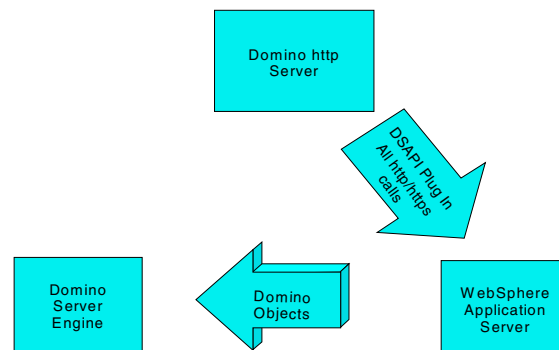


Figure 123. Connectivity between WebSphere and Domino

Thus any request for a Domino object, such as a form, agent, or document, will first be sent to WebSphere and then returned to Domino for processing. The response will flow back over the same path. This behavior is consistent with the design intent of the DSAPI plug-in to provide a low-level interface prior to any Domino processing. This behavior is transparent to the browser user.

2. IIS 4.0 and WebSphere

We found that IIS 4.0 decided, based on its configuration, which requests should be forwarded to WebSphere through the iis20.dll plug-in. In addition, requests for Domino objects were forwarded through the Domino plug-in (niisextn.dll) for processing by Domino. Finally, requests for html files in the file system were handled by IIS itself. Thus either Domino (for Domino objects) or WebSphere (for servlets and jsp's) or neither (for HTTP files from the file system) are called. The overall flow is shown in Figure 124.

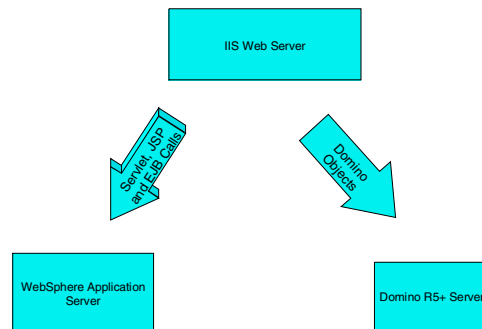


Figure 124. Connectivity between IIS, Domino and WebSphere

3. Both Domino R5 and IIS 4.0 and WebSphere

It is possible to run both IIS and Domino R5's HTTP stacks on the same host, provided they do not bind to the same port. We configured Domino to use port 8080 for HTTP for this test. We also had to define a virtual host for WebSphere to recognize calls from port 8080. This is done by opening the WebSphere Administrative Console, selecting the **Topology** tab and selecting **default_host** and then the **Advanced** tab. The new virtual host (referred to as an "alias") can be added to the bottom of the list of aliases. In our test example we added:

```
mjollner:8080
mjollner.lotus.com:8080
```

Note that the *exact* host name to be used in the URL must be present in the list for WebSphere to recognize it. Once the **Apply** button is pressed, you are prompted to stop and restart the default server for the changes to be effective.

When we did this, we could forward requests to either Domino (using port 8080) or IIS; using either HTTP stack we could also access WebSphere. Finally, we could still access Domino objects either through IIS or, of course, through the Domino HTTP stack. In summary:

- Requests to the IIS HTTP stack worked as expected:
 - Html files were returned from the file system by IIS.
 - Requests to WebSphere (for example, servlets) were routed to the WebSphere iis20.dll plug-in for processing and were returned by IIS.
 - Requests for Domino objects were routed to the niisextn.dll Domino plug-in for processing and were returned by the IIS HTTP stack
- Requests to the Domino R5 HTTP stack also worked as expected:
 - All requests were forwarded to WebSphere through the DSAPI plug-in for processing.
 - WebSphere handled requests intended for it (for example, for servlets) and returned all other requests to the Domino server running on port 8080.

This topology is shown in Figure 125.

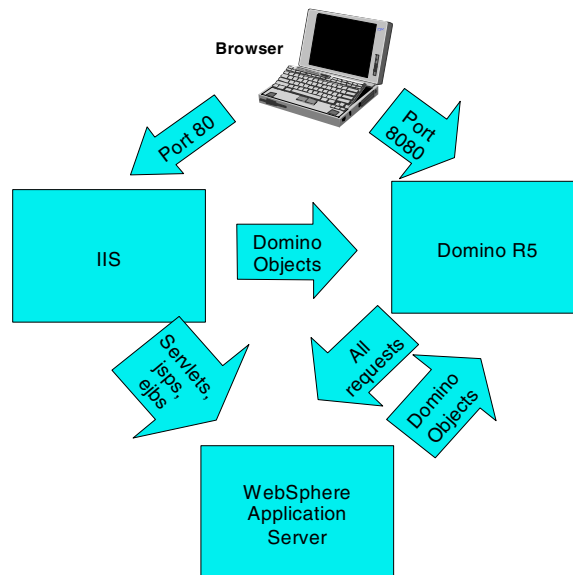


Figure 125. Running both Domino and IIS HTTP stacks with WebSphere

This configuration would not be used in a production environment but does illustrate how the products can co-exist and how the data flows between them.

B.2 Tracing Domino and WebSphere

We found it necessary to increase the default logging level of each product in order to capture detailed behavior and to investigate situations where the products did not perform as expected. This is not intended to be an exhaustive list of the options available (which are documented to some extent with the products), but a selection of techniques which we found useful.

We used the traces to assist us in documenting the above topology examples.

B.2.1 Tracing the Domino LDAP task

We found it necessary to produce detailed traces of requests to the Domino LDAP server and its responses. It is possible to get a general picture of LDAP behavior by issuing the command:

```
Show Stat LDAP
```

from the Domino console. The results will give a summary view of LDAP activity as shown in Figure 126.

```
> sh stat ldap
LDAP.Anonymous LDAP Connections = 16
LDAP.Sessions.Accept.Queue = 0
LDAP.Sessions.Active = 0
LDAP.Sessions.Inbound.BytesReceived = 15,064
LDAP.Sessions.Inbound.BytesSent = 72,629
LDAP.Sessions.Inbound.non-SSL = 57
LDAP.Sessions.Inbound.Total = 57
LDAP.Sessions.Peak = 2
LDAP.Sessions.Threads.Busy = 1
LDAP.Sessions.Threads.Idle = 0
LDAP.Sessions.Total = 57
LDAP.Simple LDAP Connections = 41
LDAP.Total LDAP Connections = 57
LDAP.Total LDAP Search Entries Returned = 48
LDAP.Total LDAP Searches = 96
```

Figure 126. LDAP output from a "show stat LDAP"

However, this summary view is of limited help to trace events because it simply accumulates totals from the time of the LDAP server start. (If you issue the console command immediately after the LDAP server is started and before there have been any LDAP requests, it will return “LDAP not found in stat table” rather than the lines in Figure 126 with zero totals).

For more detailed tracing we used the following two NOTES.INI file variables:

- **LDAPdebug=1**

This generates a trace of all calls and the results returned to the caller. The output is written to the console but can be captured to a text file by using the “Debug_Outfile” variable in the NOTES.INI file. For example:

debug_outfile=d:\Domino\Data\ldaptrace.txt

- **LDAPDebug=16**

This will display detailed hexadecimal traces on the Domino console. It is somewhat overwhelming but may be needed when you require a bit-by-bit trace of LDAP activity.

It is, of course, possible to run with both variables set.

For example, we set the variable LDAPDebug=16 and issued the command

```
ldapsearch -v -h mjollner.lotus.com "uid=WASAdmin"
```

from a command line. The resulting query was returned to the command prompt as shown in Figure 127 on page 222.

```
D:\Domino>ldapsearch -v -h mjollner.lotus.com "uid=WASAdmin"
ldap_open( mjollner.lotus.com, 389 )
filter pattern: uid=WASAdmin
returning: ALL

*** Filter is: (uid=WASAdmin) ***
CN=WebSphere Administration,O=ITSO
cn=WebSphere Administration
shortname=WASAdmin
uid=WASAdmin
mail=WebSphereAdministration@Lotus.com
objectclass=top
objectclass=person
objectclass=organizationalPerson
objectclass=inetOrgPerson
objectclass=dominoPerson
certificate=03003702 6585379C 06G01622 G002C160
givenname=WebSphere
sn=Administration
maildomain=ITSO
1 matches
```

Figure 127. Example of LDAPsearch command and response

This query produced the output on the Domino console illustrated in Figure 128 on page 223. Note that the output is detailed, but the text list on the right of the display helps to show the information being returned to the requestor. We truncated the output for clarity in this display.

```

06/28/2000 11:39:00.67 AM [01DE:0002-0212] LDAP CIServ CreateListenerTask> Listen on Port 389
> 06/28/2000 11:39:01.62 AM [01DE:0003-0200] LDAP CIServ ListenerTask> Listener task Single-Endpoint)
started
> 06/28/2000 11:39:10.88 AM [01DE:0003-0200] LDAP CIServ Listen> Connection Accepted on Port 389 for
Session 23D30006
06/28/2000 11:39:10 AM LDAP Server: 9.95.34.10 connected
06/28/2000 11:39:10.93 AM [01DE:0004-016C] LDAP> Scope: SUBTREE
06/28/2000 11:39:10.93 AM [01DE:0004-016C] LDAP> Dereference Aliases: 0
06/28/2000 11:39:10.93 AM [01DE:0004-016C] LDAP> TimeLimit: 15
06/28/2000 11:39:10.93 AM [01DE:0004-016C] LDAP> SizeLimit: 0
06/28/2000 11:39:10.94 AM [01DE:0004-016C] LDAP> Attributes to return: ALL
06/28/2000 11:39:10.94 AM [01DE:0004-016C] LDAP> Base:
06/28/2000 11:39:10.94 AM [01DE:0004-016C] LDAP> Filter: (uid=WASAdmin)
03B570C8: 30 82 06 18 02 01 02 64 82 06 11 04 22 43 4E 3D '0.....d...."CN='
03B570D8: 57 65 62 53 70 68 65 72 65 20 41 64 6D 69 6E 69 'WebSphere Admini'
03B570E8: 73 74 72 61 74 69 6F 6E 2C 4F 3D 49 54 53 4F 30 'stration,O=ITSO0'
03B570F8: 82 05 E9 30 20 04 02 63 6E 31 1A 04 18 57 65 62 '...i0 ..cnl...Web'
03B57108: 53 70 68 65 72 65 20 41 64 6D 69 6E 69 73 74 72 'Sphere Administr'
03B57118: 61 74 69 6F 6E 30 17 04 09 73 68 6F 72 74 6E 61 'ation0...shortna'
03B57128: 6D 65 31 0A 04 08 57 41 53 41 64 6D 69 6E 30 11 'mel...WASAdmin0.'
03B57138: 04 03 75 69 64 31 0A 04 08 57 41 53 41 64 6D 69 '..uidl...WASAdmi'

```

Figure 128. Output returned to Domino Console using LDAPDebug=16 ini variable

Using the NOTES.INI variable Ldapdebug=1 and the command shown in Figure 127 on page 222, we got the trace output to the Domino console shown in Figure 129 on page 224. We found this level of tracing adequate for our purposes. Using both variables set in the NOTES.INI file results in only the results returned for Ldapdebug=1 being displayed.

```

12:00:52 PM  LDAP Server: 9.95.34.10 connected
LDAP> BERGetTag State
LDAP> BERGetLeadingLengthByte State
LDAP> BERGetNext State
LDAP> Bind State
LDAP> Successful bind for Anonymous User
LDAP> Return Result State
LDAP> InitForSearch
LDAP> StateReturnResult returning resultCode 0 (Success)
LDAP> SendBufferFree
LDAP> BERGetTag State
LDAP> BERGetLeadingLengthByte State
LDAP> BERGetNext State
LDAP> Search State
LDAP>     Scope: SUBTREE
LDAP>     Dereference Aliases: 0
LDAP>     TimeLimit: 15
LDAP>     SizeLimit: 0
LDAP>     Attributes to return: ALL
LDAP>     Base:
LDAP>     Filter: (uid=WASAdmin)
LDAP> Searching in database D:\Domino\Data\names.nsf ...
LDAP>   Type of search: View Search
LDAP>     ... Opening $Users
LDAP>     ... Searching entries for a filter 'uid = WASAdmin' in
$Users
LDAP> GetSearchEntry State
LDAP>   Found matching entry, Note ID: 3890
LDAP> SendSearchEntry, sending entry CN=WebSphere
Administration,O=ITSO
LDAP> SendBufferFree
LDAP> GetSearchEntry State
LDAP> Search State
LDAP> Hits:1    LDAP> Return Result State
LDAP> InitForSearch
LDAP> StateReturnResult returning resultCode 0 (Success)
LDAP> SendBufferFree
LDAP> BERGetTag State
LDAP> BERGetLeadingLengthByte State
LDAP> BERGetNext State
LDAP> UnBind State
12:00:52 PM  LDAP Server: 9.95.34.10 disconnected

```

Figure 129. Trace output using LDAPDebug=1 in the NOTES.INI variable

B.2.2 Tracing in WebSphere

There are a number of tracing tools in WebSphere. Unfortunately, there is not a single place to adjust the logging; instead it is necessary to update some items in the WebSphere Administrative Console and to change others in text configuration files, or both.

B.2.2.1 Changing WebSphere's default trace level

This change requires you to edit the bootstrap.properties file, by default located in the \WebSphere\AppServer\properties directory.

Edit the file and navigate to the line beginning with:

```
#Set the common native web-server plugins log level
```

```
##
# Set the common native web-server plugins log level
# values of ose.native.log.level may be a combination of
# TRACE INFORM ERROR WARNING
#
#ose.native.log.level=ERROR|WARNING
ose.native.log.level=ERROR|WARNING|TRACE|INFORM
##
# Set the plugin specific native web-server plugins log level
# values of ose.native.log.level may be a combination of
# TRACE INFORM ERROR WARNING
#ose.plugin.log.level=ERROR|WARNING
ose.plugin.log.level=ERROR|WARNING|TRACE|INFORM
```

Figure 130. Increasing the trace level of the WebSphere Application Server

Change the text as shown in Figure 130 to add the variables **TRACE|INFORM** to the lines **ose.native.log.level** and **ose.plugin.log.level**. In our example, we made copies of the original lines and commented them out. Once these changes are made, the WebSphere administrative server (and the administrative console) have to be stopped and restarted to make the changes effective. Note that the size of the trace files will grow rapidly since every event in the WebSphere server will be traced. Therefore, you will want to use this level of logging when investigating a specific problem rather than use it as your usual logging level.

The trace logs produced will be written to the logs directory under the WebSphere installation directory (\WebSphere\AppServer by default). The file names have a description portion followed by a timestamp to help identify them by type and date as shown in Figure 131 on page 226.

```

349 adminserver_native.log.was-oop.Mon-Jun-26-10.24.54-2000
    0 default_server_native.log.was-oop.Mon-Jun-26-09.25.06-2000
    0 default_server_native.log.was-oop.Mon-Jun-26-10.27.02-2000
146,007 default_server_stderr.log
644,318 default_server_stdout.log
203,252 trace.log.domino.Fri-Jun-23-18.21.04-2000
1,714,101 trace.log.iis.Wed-Jun-14-09.53.52-2000
515,777 tracefile
348 wasdb2.log

```

Figure 131. WebSphere Log file types

You may have different log files in your log directory depending on what other options you have specified. You will almost certainly have many more than the above display (the list from which Figure 131 was extracted had 195 log files). Most of the information relating to Domino and IIS would be found in the trace.log.domino... and trace.log.iis... log files respectively. The most useful information to us was in the trace.log.domino... log files. For example, Figure 132 shows a (partial) listing of WebSphere returning a call to Domino for processing.

```

Trace - sysmgmt_is_servlet_uri : queue not found for uri /
Trace - ose_is_servlet_uri(): FALSE
Trace - service_exit return_code: 0
Trace - In Authenticate
Trace - ws_is_request_protected: pose_data->init_complete=1
Trace - sysmgmt_sec_for_uri: host=mjollner : port=8080 :
uri=/homepage.nsf/8d8d42cca06351c98525673100664d8b/$Body/0.84?OpenElement&FieldElemFormat=gif
Trace - uri_len=-1 root_len=-1 mime_len=-1
Trace - ws_rule_find_sec: secparm not found
Trace - sec_is_request_protected for mjollner : 8080 :
/homepage.nsf/8d8d42cca06351c98525673100664d8b/$Body/0.84?OpenElement&FieldElemFormat=gif

```

Figure 132. Partial listing of Domino trace log in WebSphere log directory

It is possible from the (entire) log to examine each of the header elements and responses in detail. This can be useful (for URLs that require WebSphere action) to determine whether or not expected data is being generated and returned to Domino. An example could be a cookie or servlet output. The Trace logs for IIS are similar, but only URLs for WebSphere objects (for

example, servlets) are listed since IIS does not sent non-WebSphere requests to it.

The default_server_stdout.log and default_server_stderr.log files are the standard output and standard error files written by the Default Server. These file names can be configured on the **General** tab page of the Default Server topology view. These files will contain the output written by servlets, JSPs and EJBs via the System.out and System.err objects.

B.2.2.2 Tracing the WebSphere Administrative Console

This is run from a JVM initiated from a command prompt. You can run the Administrative Console in “debug” mode, but by default the messages are written to the command prompt window. You can capture the output by redirecting it to a file using the following steps:

1. Open a command prompt window and increase its buffer size to at least 200 lines (from the properties panel for the command prompt).
2. Change the working directory to \WebSphere\AppServer\bin
3. Run the command:

```
adminclient debug > console_trace.txt
```

This will capture any error or trace messages from the Administrative Console to the “console_trace.txt” file for later inspection.

B.2.2.3 Security tracing in WebSphere Application Server

The first step is to edit the admin.config file found in the \WebSphere\AppServer\bin directory by default. Ensure that you have a backup of this file since the WebSphere server will be inoperable if you corrupt it.

Open the file in a text editor and locate the bottom of the file. Add the following lines to the end of the file and save it:

```
com.ibm.ejs.sm.adminServer.traceOutput=x:/WebSphere/AppServer/logs/  
  admin_security_trace.txt  
com.ibm.ejs.sm.adminServer.traceString=com.ibm.ejs.security.*=all=enabled
```

Note that both strings must be on *single* lines. The WebSphere server will have to be restarted before these settings become effective. However, before doing this there are further settings that need to be altered in the Administrative Console:

1. Start the console if it is not already running. Open the **Topology** tab, and select the application server under test (for example, the Default Server) and open the **Advanced** tab. In the field "Trace specification" enter the string:

```
com.ibm.ejs.security.*=all=enabled
```

2. In the field "Trace Output File" enter a path and filename of a file to receive security output (for example, `x:/WebSphere/AppServer/logs/app_server_security_trace.txt`) where "x" is the drive you installed WebSphere on. Note that the separators are *forward* slashes in this and the preceding lines.

3. Press the **Apply** button.

4. Stop and restart the WebSphere Server and the Administrative Console. The WebSphere Server can be stopped and restarted in one operation by selecting the hostnode and right-clicking to display a context menu. Clicking on **Restart** will stop and restart the WebSphere Server and stop the Administrative Console (which must be restarted manually after the WebSphere Server has restarted).

In our experience, this simply captured routine security events such as the analysis of the security configuration and servlet instantiation messages. However, if there had been any security exceptions during the log period, they would have been captured in this log.


```

Trace - sysmgmt_sec_for_uri: host=mjollner : port=8080 :
uri=/homepage.nsf/8d8d42cca06351c98525673100664d8b/$Body/0.84?OpenElement&FieldElemFormat=gif
Trace - uri_len=-1 root_len=-1 mime_len=-1
Trace - ws_rule_find_sec: secparm not found
Trace - sec_is_request_protected for mjollner : 8080 :
/homepage.nsf/8d8d42cca06351c98525673100664d8b/$Body/0.84?OpenElement&FieldElemFormat=gif
0 - default_host : : NULL
Trace - sec_is_request_protected indicated 0
Trace - Request is not protected
Trace - uri_len=-1 root_len=-1 mime_len=-1
Trace - ws_rule_find_queue: queue name was not found
Trace - sysmgmt_is_servlet_uri : queue not found for uri
/homepage.nsf/8d8d42cca06351c98525673100664d8b/$Body/0.84?OpenElement&FieldElemFormat=gif
Trace - ose_is_servlet_uri(): FALSE
Trace - service_exit return_code: 0
Trace - In Authenticate
Trace - ws_is_request_protected: pose_data->init_complete=1
Trace - sysmgmt_sec_for_uri: host=mjollner : port=8080 :
uri=/servlet/snoop
Trace - uri_len=13 root_len=-1 mime_len=-1
Trace - sec_is_request_protected for mjollner : 8080 : /servlet/snoop
0 - default_host : /servlet/snoop : NULL
Trace - sec_is_request_protected indicated 0
Trace - Request is not protected
Trace - uri_len=13 root_len=-1 mime_len=-1
Trace - sysmgmt_is_servlet_uri : uri /servlet/snoop is for queue
ibmoselink
Trace - ose_is_servlet_uri(): TRUE
Trace - ose_request_stub.version: 1
Trace - ose_request_stub.scheme: http
Trace - ose_request_stub.method: GET
Trace - ose_request_stub.protocol: HTTP/1.1
Trace - ose_request_stub.req_uri

```

Figure 133. Security trace from the WebSphere Application Server

Appendix C. Additional Web material

Additional Web material is referenced in this Redbook and can be found on the IBM Redbooks Web site. The material is shown in Table 19.

Table 19. Additional Web material

File name	Description
5955-jar.zip	Zipped file containing sg245955.jar and RedDomTestEJB.jar with all the VA Java examples from Chapter 4 and 5.
5955-jsp.zip	Zipped file containing the JavaServer Page examples from Chapter 4.
5955-nsf.zip	Zipped file containing the Domino database banking.nsf with the sample forms and views used in Chapter 5.

C.1 How to get the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG245955>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

Appendix D. Special notices

This publication is intended to help architects and developers to understand how Domino and WebSphere integrate from a technical angle. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server or the Domino server family. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Application Server, and Lotus Domino R5 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee

that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM ®	AIX
AS/400	CICS
DB2	DB2 Universal Database
MQSeries	S/390
SecureWay	VisualAge
WebSphere	

The following terms are trademarks of Lotus Development Corporation in the United States and/or other countries:

ILotus ®	Lotus Notes ®
Lotus Domino	LotusScript ®
Domino Workflow	Domino.Doc
Lotus SmartSuite ®	Lotus QuickPlace
People Places and Things ®	Lotus Sametime
SUPER.HUMAN.SOFTWARE	

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Københavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 241.

- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429
- *Lotus Domino Release 5.0: A Developer's Handbook*, SG24-5331, Lotus part number CC7EDNA
- *Lotus Sametime Application Development Guide*, SG24-5651, Lotus part number CT7AKNA
- *Using Domino Workflow*, SG24-5963, Lotus part number CT6GNML
- *Using VisualAge for Java to Develop Domino Applications*, SG24-5424, Lotus part number CT6ENNA
- *Connecting Domino to the Enterprise Using Java*, SG24-5425, Lotus part number CT6EMNA
- *Lotus Domino R5.0 Enterprise Integration: Architecture and Products*, SG24-5593, Lotus part number CT6QUNA
- *Performance Considerations for Domino Applications*, SG24-5602, Lotus part number CT7V6NA
- *Lotus Notes and Domino R5.0 Security Infrastructure Revealed*, SG24-5341, Lotus part number CT6TPNA
- *WebSphere Application Servers: Standard and Advanced Editions*, SG24-5460
- *WebSphere V3 Performance Tuning Guide*, SG24-5657
- *IBM WebSphere Performance Pack: Caching and Filtering with IBM Web Traffic Express*, SG24-5859
- *IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher*, SG24-5858

- *IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System*, SG24-5857
- *Application Server Solution Guide Enterprise Edition: Getting Started*, SG24-5320
- *IBM WebSphere Transcoding Publisher V1.1: Extending Web Applications to the Pervasive World*, SG24-5965
- *Linux Web Hosting with WebSphere, DB2, and Domino*, SG24-6007
- *Developing an e-business Application Using Lotus Domino for AS/400*, SG24-6052
- *WebSphere Application Server Enterprise Edition Component Broker 3.0 First Steps*, SG24-2033

E.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

E.3 Other resources

These publications are also relevant as further information sources:

- *WebSphere Application Server Support page* - contains technical notes, troubleshooting help, service packs, E-fixes and more available online at <http://www.ibm.com/software/webservers/appserv/support.html>

- *Lotus Knowledge Base* - contains Tech Notes and Papers, available online at
<http://support.lotus.com/>
- *Hypertext Transfer Protocol -- HTTP/1.0 RFC 1945 including Basic Authentication*, available online at
<http://www.cis.ohio-state.edu/htbin/rfc/rfc1945.html>
- *Hypertext Transfer Protocol -- HTTP/1.1 RFC 2616*, available online at
<http://www.cis.ohio-state.edu/htbin/rfc/rfc2616.html>
- *Cookies: RFC 2109 HTTP State Management Mechanism*, available online at
<http://www.cis.ohio-state.edu/htbin/rfc/rfc2109.html>
- A good source for Meta Tag information, available online at
<http://vancouver-webpages.com/META/>
- *AFS enterprise file system FAQ*, available online at
<http://www.angelfire.com/hi/plutonic/afs-faq.html>
- *Expanded Command Caching in Domino 4.61*, Article in Iris Today, available online at
<http://www.notes.net>

E.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- <http://ibm.com/websphere/> Entry point to information about the IBM WebSphere software platform for e-business
- <http://www.lotus.com/developer/> Lotus' primary destination for the latest developer information and resources. Contains articles about new and current technologies along with relevant tips and techniques to help you build dynamic collaborative e-business applications.
- <http://notes.net/> Notes.net from Iris - the developers of Notes and Domino - is a technical Web site with discussion forums, documentation and the Webzine Iris Today with many good articles about technical details of Domino.
- <http://ibm.com/developer/> The IBM developerWorks Web site is designed for software developers, and features links to a host of developer tools, resources, and programs.

- <http://support.lotus.com/> Lotus Support's Web site - Search using keywords or browse the Lotus Knowledge Base and locate helpful and informative tech notes and technical papers for the entire Lotus Product family. This source of information contains the latest technical information updated hourly.

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** ibm.com/redbooks

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name
------------	-----------

Company

Address

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

<input type="checkbox"/> Invoice to customer number	
---	--

<input type="checkbox"/> Credit card number	
---	--

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

A

- Admin console for WebSphere 37
 - assign permissions 161
 - configure a Web application 171
 - configure an enterprise application 172
 - disable security 150
 - enable security using Domino Directory 150
 - starting default server 37

C

- Caching
 - ? and ! in URLs 198
 - Domino examples 195
 - R5 View Applet considerations 200
- CLASSPATH
 - adding JdbcDomino.jar 76
 - extracting archives 132
 - for IBM JDK 20
 - for Notes.jar 66
 - in WebSphere 66
 - system versus user variable 21
 - used by WebSphere fix pack 34
- Collaborative commerce 1
- Cookie
 - common cookie 3
 - session based authentication 189

D

- DB2
 - data source in WebSphere 69
 - first steps 15
 - installation 12
 - JDBC driver 67
 - SAMPLE database 15
- Directory sharing 146
- Domino
 - ? or ! in URLs 198
 - access via JDBC 75
 - accessing from a JSP 94
 - Account form v1 109
 - Account form v2 112
 - agents and servlets 50
 - and Network Dispatcher 191
 - and Web Traffic Express 193
 - authentication review 187

- caching examples 195
- calling JSPs 115
- configuring IIS as Web server 209
- different ways to access EJB 143
- directory assistance 162
- Domino Directory and WebSphere 146
- external LDAP directory 162
- httpd.cnf 39
- installing administration client 30
- installing Domino server 21
- installing WebSphere DSAPI plug-in 40
- invoking EJBs 118
- invoking servlets 105
- NCSOW.jar 62
- Notes.jar 61
- passing data to servlets 108, 111
- protecting a Web application 169
- server document: Java servlet support 43
- session based authentication 189
- tracing the LDAP task 220
- TransferOrder form 123
- Transfers view 123
- use from EJB 96
- using basic authentication 165
- using EJB from agent 123, 135
- using EJB via servlet 141
- using IIS as Web server 45
- verifying WebSphere servlet support 42
- Domino Internet Cluster Manager 185

E

- e-business
 - IBM e-business foundation 5
- EJBs
 - accessing Domino 96
 - Account EJB 67
 - architecture 54
 - Bean Managed Persistence 53
 - Container Managed Persistence 53
 - creating an EJB container 70
 - deploying from VA Java 100
 - deploying in WebSphere 67
 - entity bean 53
 - home class 120
 - Home interface 55
 - introduction 52
 - invoking from Domino 118

- Object or remote interface 55
- session beans 53
- steps in using 56
- Transfer EJB 67
- VA Java enterprise edition 97

H

- httpd.cnf 39

I

- IBM JDK
 - installation 18
- IBM SecureWay Directory 155
 - administration interface 156
 - as external directory for Domino 162
 - Directory Management Tool 156
 - ldif2db utility 156
 - sample.ldif 156
- IIS 205
 - Configuring for Domino 209
 - configuring for WebSphere 214

J

- Java packages
 - DeployedTransfer.jar 132
 - ejs.jar 119
 - extracting archives 132
 - iioptools.jar 119
 - java.rmi 120
 - java.util 120
 - javax.ejb 120
 - javax.naming 120
 - javax.rmi 120
 - JdbcDomino.jar 76
 - NCSOW.jar 62
 - Notes.jar 61, 63
 - RedDomTestEJB.jar 98
 - ujc.jar 119
- JDBC
 - DB2 driver 67
 - driver for Domino R5 75
- JSPs
 - 0.91 support 86
 - 1.0 support 86
 - accessing Domino 94
 - called from Domino 115
 - deploying example files 74

- introduction 50
- passing data to 116
- ReadView.jsp sample 83
- request parameters 83
- scriptlet code 84
- SendMail.jsp sample 94
- use of LotusScript 3

L

- LDAP
 - Base DN 163
 - configuring Domino LDAP 147
 - default schema 153
 - LDIF file 156
 - suffix 163
 - Tracing in Domino 220
- Limits
 - DB2 user ID 9
 - passing data via URL 110

N

- NOTES.INI
 - JavaUserClasses 119, 131, 132
 - LDAPDebug 221
 - ServerTasks, LDAP 149

P

- PATH
 - library path in WebSphere 66
 - setting for Domino server 30

R

- RMI
 - accessing EJBs 118
 - RMI registry 139
 - using to access EJB 134

S

- Security
 - basic authentication 151, 165
 - common cookie 3
 - enable/disable in WebSphere 150
 - method groups in WebSphere 174
 - protecting a Web application 169
- SecurityException
 - in Domino Java agent 131
- Servlets

- aliases in WebSphere 108
- and Domino Web agents 50
- CreateAccountMail sample 87
- CreateAccountPost sample 113, 172
- introduction 49
- invoking EJBs 101
- invoking from Domino 105
- passing data by form 111
- passing data by URL 108
- ReadNames sample 76
- sendRedirect method 114
- snoop servlet 42, 108
- verifying WebSphere servlet support 42

U

User ID

- length limit for DB2 9
- with administration rights 9

V

VisualAge for Java

- deploy EJB 100
- deploying servlets to WebSphere 73
- enterprise edition 97
- home class for EJB 120
- IBM Account example 65
- IBM WebSphere Test Environment 3.0 61
- Lotus Domino Java library 5.0 62
- setting up 59

W

Web server plug-in

- configuring domino5.dll in Domino 40
- differences in behaviour 217
- different behaviour 45
- installlation 31

WebSphere

- adding NCSOW.jar to class path 92
- admin console, See Admin console for WebSphere 37
- application components 47
- data source definition 69
- default server 37
- deploying example EJBs 67
- disable security 150
- EJB container 70
- EJB deployment 100

- enable security using Domino Directory 150
- enterprise application 171
- installing advanced edition 30
- installing service upgrade 34
- JDBC driver 67
- JSP version support 86
- library path in admin.config 66
- method groups 174
- protecting a Web application 169
- redeploy EJB 102
- servlet aliasing 108
- starting the AdminServer service 36
- trace level, changing 225
- using another LDAP directory 155
- using basic authentication 167
- using Domino Directory 150
- using IIS as Web server 45, 214
- Web application 171
- WebSphere Performance Pack 179
 - AFS Enterprise file system 184
 - Content Based Routing 183
 - Network Dispatcher 180
 - Web Traffic Express 181

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at ibm.com/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number	SG24-5955-00
Redbook Title	Domino and WebSphere Together
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
Please identify yourself as belonging to one of the following groups:	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<input type="checkbox"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. ibm.com/privacy/yourprivacy/



Domino and WebSphere Together

(0.5" spine)

0.5" <-> 0.875"

250 <-> 459 pages



Redbooks

Domino and WebSphere Together

Installation and setup

Access Domino objects from WebSphere

Use servlets and EJBs from Domino

Lotus Domino, IBM WebSphere, and their related products can be used to deliver new value to customers in the form of "collaborative commerce services." In this redbook we explain how you can use the combined capabilities of Domino and WebSphere to provide a complete and integrated platform for collaborative commerce.

Using examples and scenarios, we show you how to install and configure Lotus Domino R5 and IBM WebSphere 3.02 on Windows NT. We then look at how the components in a WebSphere Web application (servlets, JavaServer Pages and Enterprise JavaBeans) can work together with the functionality of Domino.

We also investigate authentication and authorization issues and guide you through the steps needed to protect a combined Domino and WebSphere application. Finally, we discuss options for scalability and redundancy when using Domino with its Internet Cluster Manager or the WebSphere Performance pack.

This redbook will help IT architects and developers at business partners, solution developers and customers to understand the technical integration of Domino and WebSphere, so they can effectively exploit the combined strength of these products.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-5955-00

ISBN 0738416428